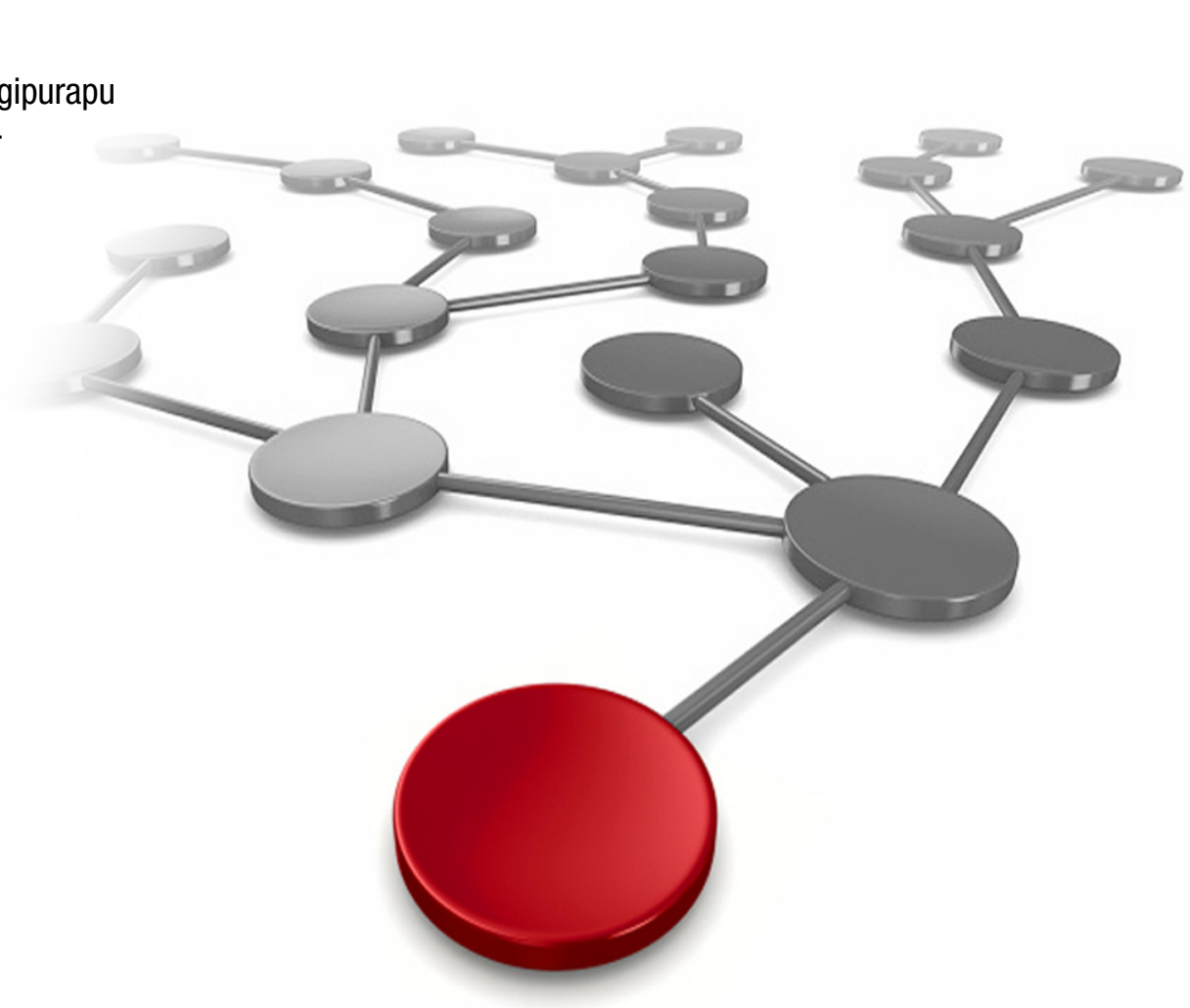
Redbooks
ibm.com/redbooks

Delivering SAP with IBM Bob

Dharma Atluri
Tuhin Bhaduri
Shubhangi Joshi
Phillip Gerrard
Meher Charan Vangipurapu
Mohammad Zubair



IBM

Introduction

This document provides practical guidance for IBM consultants, SAP practitioners, and enterprise architects on using IBM Bob, IBM's AI-native engineering assistant, to accelerate and govern SAP development. It covers the full spectrum from initial setup and connectivity through governed code generation, workflow automation, and integration with the broader SAP and enterprise tooling ecosystem.

The focus is on real-world applicability: every section is grounded in hands-on implementation experience across SAP landscapes including S/4HANA, SAP BTP, SAP Integration Suite, and ECC. Readers can find concrete configuration steps, example prompts, architectural guidance, and worked examples that can be adapted directly to client engagements.

For more information and disclaimers about the information and code examples provided in this publication, see the notices section of this document.

Who should read this document

This document is intended for the following users:

- **SAP Developers and ABAP Engineers** who want to accelerate code generation, refactoring, testing, and documentation using AI-assisted tooling integrated directly into their development environment
- **SAP Architects and Solution Architects** who need to understand how IBM Bob fits into the broader SAP and enterprise architecture, how governance controls are applied, and how to define standards that the tool enforces across teams
- **IBM Consultants and Project Leads** who are preparing to introduce IBM Bob on a client engagement and need a comprehensive reference covering both the technical setup and the delivery methodology
- **DevOps and Platform Engineers** who are responsible for connecting IBM Bob to SAP systems, CI/CD pipelines, and external enterprise tools such as ServiceNow, GitHub, and Tricentis

You should have working knowledge of SAP development concepts (ABAP, BTP, or SAP Integration Suite), familiarity with Visual Studio Code, and a basic understanding of AI coding assistants. No prior experience with IBM Bob is required.

Section overview

Using IBM Bob for SAP introduces IBM Bob and makes the case for its adoption in SAP environments. It covers the market forces driving AI-native engineering, what IBM Bob is and how it differs from generic coding assistants, a comparison with SAP's own Joule 4 Developers tooling, and key development approaches, including Spec Driven Development and Vibe Coding, that shape how the tool is used most effectively.

Connecting IBM Bob to SAP provides the technical foundation for integrating IBM Bob with SAP systems. It covers the available connectivity approaches (ABAP Remote FS Extension, ABAP ADT Extension, and the SAP ADT MCP Server), deployment models, architectural considerations for S/4HANA, ECC, and SAP BTP, and a complete walkthrough for governed AI-assisted development using abapGit, including step-by-step configuration, transport management, and GitHub connectivity.

Working with IBM Bob establishes the operational framework for using IBM Bob in enterprise SAP projects. It covers IBM Bob's built-in modes (Agent, Plan, and Ask) and how to author custom modes for SAP domains, the full permission model, and the Dual-Engine Governance model of Rules and Skills. This section also addresses centralized configuration and distribution for teams, SAP Clean Core and ABAP Cloud standards enforcement, naming conventions, coding standards, and Spec-Kit mode for spec-driven delivery.

Development with IBM Bob provides deep, hands-on coverage of IBM Bob across specific SAP development domains. Topics include ABAP code generation tips and best practices, live code generation via the ADT and ABAP FS extensions, SAP BTP UI5 and CAP development, SAP Cloud Integration (CPI) Groovy script generation and governed architecture, and comprehensive guidance on generating SAP project documentation including functional specifications, technical specifications, and business process procedures.

Integrations and Operation covers IBM Bob's integration with the enterprise tooling ecosystem beyond SAP. It explains how the Model Context Protocol (MCP) enables a single governed interface to multiple systems, then provides detailed integration guides for ServiceNow (change and incident management), GitHub and CI/CD pipelines, Tricentis Tosca (SAP test automation), SAP Signavio and LeanIX (process and enterprise architecture intelligence), and SAP BTP administration. Each integration includes architecture diagrams, authentication configuration, and practical worked examples.

Getting Started provides a hands-on environment setup guide for all three primary SAP development tracks, ABAP, SAP BTP UI5/CAP, and SAP Integration Suite (CPI). It walks through prerequisites, extension installation, MCP server registration, governed rules and skills configuration, and end-to-end verification. The section closes with a Crawl–Walk–Run adoption roadmap that maps the progression from individual developer productivity through enterprise-scale distributed delivery.

Governance and Scale addresses the governance disciplines required to operate IBM Bob responsibly in production SAP engagements. It covers the layered guardrail model (data access, action, model behavior, and audit boundaries), security posture, human-in-the-loop design patterns, and a practical framework for measuring value across immediate, project-level, and strategic dimensions. The section concludes with a 30–60–90 day action plan to move from initial adoption to embedded, scaled delivery.

Appendix A provides the complete source code for the `ZDOWNLOAD_ABAP_OBJECT` sample ABAP program, which serializes selected ABAP objects with abapGit and downloads the resulting files to a local folder so that IBM Bob can work with them. It is referenced from the Connecting IBM Bob to SAP section.

Authors

Dharma Atluri is an IBM Distinguished Engineer, Global CTO for AI-Led SAP Manage with over 25 years of experience in SAP and with IBM for over 16 years. He is a published author with SAP-Press, Packt, and Archiee delivering both technical and industry-driven books regarding the roles of AI and Cloud based journey for clients. He is also an IBM Senior Inventor with 3 granted patents and overall 18 published patents. He is an active blogger and has experience delivering global enterprise-scale transformation journeys for clients across industries. He has been actively driving the global adoption of IBM Bob for SAP engagements along with ICA Edge and other IBM Tech products.

Tuhin Bhaduri is a seasoned AI Architect and SAP transformation leader with over 22 years of experience delivering enterprise-scale innovation across global organizations. He specializes in helping clients build, adopt, and scale AI capabilities within complex enterprise landscapes, particularly across the SAP ecosystem. Currently, Tuhin supports multiple enterprise accounts in designing and implementing AI-driven business solutions that accelerate operational efficiency, intelligent automation, and digital transformation. His expertise spans Agentic AI, watsonx Orchestrate, IBM Consulting Advantage, and IBM Bob, enabling organizations to create intelligent, enterprise-ready AI solutions. Tuhin brings deep functional and technical expertise across SAP technologies, including SAP BTP (BAIP), SAP S/4HANA, SAP Signavio, and LeanIX. His extensive background in enterprise SAP development and transformation allows him to bridge business strategy with modern AI and automation capabilities, helping organizations modernize processes and unlock measurable business value. With a strong combination of architectural vision, SAP domain expertise, and AI innovation experience, Tuhin is recognized for driving impactful enterprise transformation initiatives across industries.

Shubhangi Joshi is a seasoned SAP Architect with over 20 years of experience delivering enterprise-scale SAP transformation programs. A published SAP Press author of "ABAP on Cloud," she is an expert in SAP S/4HANA, SAP Business Technology Platform (BTP), SAP Business AI, Clean Core, and cloud-native development. Passionate about innovation, Shubhangi helps organizations accelerate their digital journeys through scalable and future-ready

solutions. Her core specializations include enterprise architecture, SAP extensibility, Generative AI, Agentic AI, and modernization strategies. Known for her high energy and collaborative leadership, she is a frequent contributor to technical communities, knowledge-sharing initiatives, and enablement programs across the IBM and SAP ecosystems. Her professional mission is to bridge business challenges with emerging technologies, driving sustainable value and team growth through a customer-first mindset.

Phillip Gerrard is a Project Leader for the International Technical Support Organization working out of Beaverton, Oregon. As part of IBM® for over 20 years, he has authored and contributed to hundreds of technical documents published to IBM.com and worked directly with IBM's largest customers to implement storage solutions and resolve critical situations. As a team lead and Subject Matter Expert for the IBM Spectrum Protect support team, he is experienced in leading and growing international teams of talented IBMers, developing and implementing team processes, creating and delivering education. Phillip holds a degree in computer science and business administration from Oregon State University.

Meher Charan Vangipurapu is an Architect at IBM with 19 years of enterprise technology experience spanning NLP-based intelligent automation, SAP delivery, and Generative AI. He has served as an Ansible Architect across multiple IBM engagements covering both SAP and non-SAP landscapes, and has applied GenAI tools including IBM Bob, ICA Edge, Ansible Lightspeed, and IBM Consulting Advantage to accelerate delivery using spec-driven and vibe coding approaches. Meher currently focuses on architecting Agentic AI and intelligent automation solutions for SAP ecosystems, helping clients unlock measurable business value through modern AI-assisted development practices.

Mohammad Zubair is a Solution Architect at IBM with 19 years of industry experience, including 16 years at IBM, specializing in enterprise automation, SAP transformation, and Generative AI. He has led the Ansible automation workstream as an Architect for SAP RISE engagements, driving automation across SAP Day 0, Day 1, and Day 2 operations. He has also contributed to NLP- and AI-driven innovations through initiatives such as Ticket Analytics and Catapult Wisdom. Zubair brings strong hands-on expertise across IBM Bob, ICA Edge, Ansible Lightspeed, Ansible Automation Platform, and Agentic AI solution design. He currently leads AI-driven solution architecture and intelligent automation initiatives for SAP clients as part of IBM's AI-Led SAP Manage practice, focusing on modernizing enterprise operations through automation, Generative AI, and agentic solutions.

Feedback

This document reflects practical experience implementing these technologies in production environments. If you encounter issues not covered here, or if you have suggestions for improving the guidance, your feedback helps make this resource more valuable for the community.

Contact IBM Redbooks

Using IBM Bob for SAP

This section provides a comprehensive introduction to IBM Bob in the context of SAP development. It covers the market forces driving the need for AI-native engineering tools, what IBM Bob is and what makes it uniquely suited for enterprise SAP landscapes, the key development approaches it enables, and how it is positioned alongside the AI tools from SAP. By the end of this section, you will have the context needed to understand why IBM Bob matters for SAP and how to begin thinking about its adoption in your organization.

Introduction and overview

AI is no longer just assisting code development; it is redefining how software gets built.

This evolution also brought different challenges that the market must address at the same time:

- Pressure to cut development costs while introducing unpredictable AI expenses and to avoid compromising reliability of core systems.

- Slow and expensive modernization efforts due to manual, expertise-heavy software development lifecycle (SDLC) processes.
- Increasing dependence on a shrinking pool of experts creating delivery risk and capacity bottlenecks.
- Highly manual debugging and rework required to minimize impact on core systems.
- Most AI tools aren't enterprise ready and can't handle complex, governed codebases.
- High context-switching and "prep time" fatigue slowing down project completion.
- Compliance is a burden because manual reviews are slow and increase audit risk.

What are the most important trends to address these challenges and what are the analysts saying about these trends?

AI-Native Engineering

- Enterprises are moving beyond standalone AI agents embedded in IDEs toward AI-native engineering systems that integrate across the full software development lifecycle.
- These platforms unify planning, coding, testing, and deployment while maintaining shared context (codebase, tickets, infrastructure).
- This shift reflects a broader move from fragmented tools to end-to-end system platforms that embed AI at every stage of development.[1]

Agentic Execution

- AI is evolving from a passive assistant into agentic systems capable of planning and executing tasks autonomously. Instead of simply generating code suggestions, these systems can complete workflows such as feature development, testing, or issue resolution with minimal human intervention.
- This marks a shift in value from improving developer productivity to delivering measurable outcomes and completed work at scale.[2]

Orchestration

- As AI-generated output increases, the primary challenge becomes coordinating multiple agents, workflows, and system dependencies rather than generating code itself.
- Developers are increasingly responsible for defining goals, supervising AI execution, and ensuring system coherence.
- This elevates orchestration and task routing, agent coordination, and workflow management to a core layer of the engineering stack.[2]

Governance

- As AI systems take on business-critical engineering tasks, enterprises require robust governance frameworks to manage risk, compliance, and reliability.
- This includes capabilities such as auditability, explainability, access control, and lifecycle monitoring.
- Without these controls, organizations face barriers to scaling AI adoption in production environments, making governance and observability essential platform requirements rather than differentiators.[3]

This publication builds on the preceding trends. It provides an overview of how SAP development has evolved and where IBM Bob fits into that journey. It also examines multiple scenarios in depth, showing how IBM Bob can help you as your personal digital twin and giving you details to start executing from day 1.

Evolution in market — Need for coding tools focused on SAP

SAP development has been evolving rapidly over the last couple of years, and traditional SAP development models, often dependent on SAP GUI-centric workflows, fragmented tools, and manual issue resolution are no longer sufficient for the speed, agility, and automation expected in modern enterprise IT landscapes.

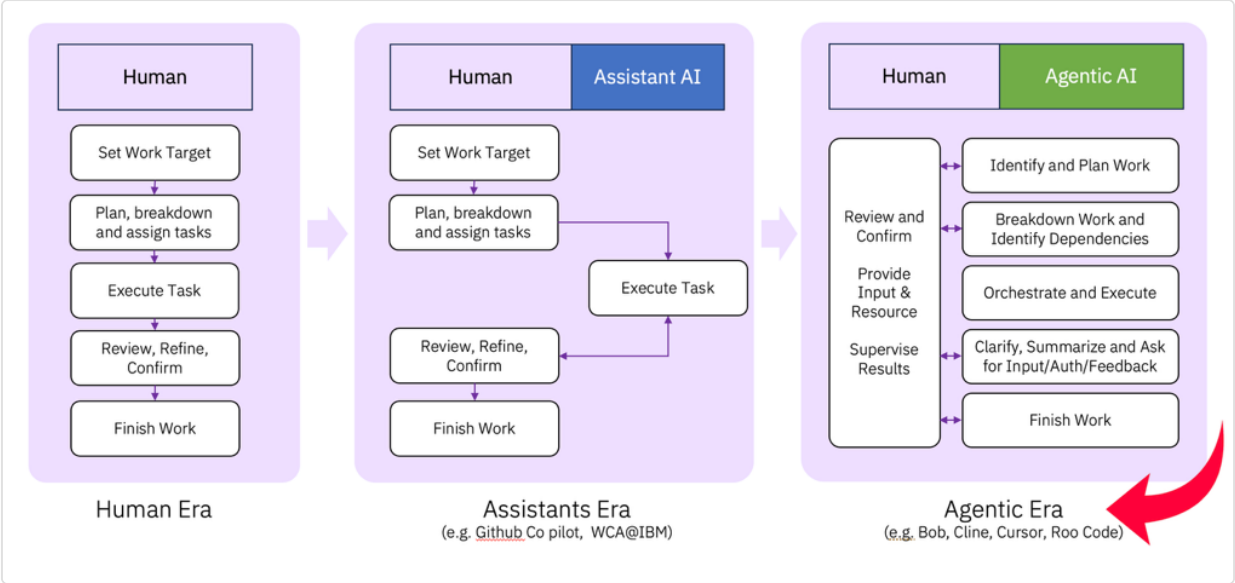


Figure: Evolution from human era to agentic AI era

To address the preceding evolution of the journey, we also need the roles to evolve rapidly, because AI-based augmentation alone cannot deliver successful and measurable outcomes.

How are the roles impacted with this evolution?

Although much of the focus is on how coding agents change the way artifacts are delivered, the following list gives a quick overview of how roles are transforming when infused with these coding agents:

- Business Analyst → Product Designer
- Industry Consultant/Project Manager → Product Manager
- Enterprise Architect → Forward Deployed Architect
- Integration Architect/App Architect → Forward Deployed Engineer
- App Developer/Quality Engineer → Applied AI Specialist
- Quality Engineer → QE – Performance and Resilience
- Application Consultant → Forward Deployment Practitioner
- BASIS/Security Consultants → AI Augmented BASIS/Security Consultants focusing on AI adoption scenarios

Options available for coding agents related to SAP

There are multiple options available in the market today that address these requirements and attempt value-driven orchestration, including the following key options:

- **Incumbent hyperscalers**: Microsoft GitHub Copilot, AWS Kiro, Google Antigravity
- **Rapidly adopted specialists**: Claude Code, Cursor, Windsurf
- **Build your own with open source**: OpenCode, OpenHands, Cline, and others
- **IBM Bob**

The following sections take a more in-depth look at IBM Bob in the context of SAP.

Overview of IBM Bob

AI-assisted development has advanced quickly, but enterprise software demands coordination, governance, and scale that most tools were not designed to handle. Most of the existing tools mentioned previously fall short because they focus on optimizing isolated tasks unrelated to enterprise workflows, and they lack context across systems when they must coordinate end-to-end complex changes.

IBM Bob is a next-generation AI-powered engineering assistant from IBM — a VS Code-native agentic IDE built on large language models (multimodel approach including those from IBM watsonx and Anthropic Claude), extended through the Model Context Protocol (MCP). It operates as a stateful, multimodal agent capable of reading, writing, linting, and handling SAP and non-SAP based coding scenarios — all from within the developer's IDE.

Unlike generic AI coding assistants, IBM Bob is purpose-built for enterprise SDLC automation, with deep SAP integration, governance controls (modes, skills, rules) and a full artifact lifecycle from requirements to production deployment.

Key attributes that define the role of IBM Bob:

Your trusted development partner your teams can rely on

IBM Bob completes tasks across the SDLC — not just code suggestions — by interpreting intent, designing structured plans, executing coordinated changes, validating outcomes, and governing delivery to produce consistent, enterprise-ready results across environments.

This provides the following benefits:

- Faster ramp-up and more consistent execution across teams
- Enterprise-ready results with validated, governed changes end-to-end
- Reduced cognitive load, clearer intent, and fewer rework cycles
- Less dependency on senior engineers through codified best practices and guardrails

Modernization that improves velocity without sacrificing quality

IBM Bob accelerates large-scale modernization while preserving system integrity, so teams can move faster without increasing technical debt, security risk, or operational fragility.

This provides the following benefits:

- Enables modernization programs to progress continuously with measurable risk reduction at each step
- Repo-wide refactoring, dependency upgrades, and framework migrations
- Automatic test regeneration and CI alignment

Cost optimization and transparency that sustains innovation

IBM Bob brings cost discipline to AI-assisted development through a simplified pricing model, centralized cost visibility, and multimodel orchestration, optimizing compute, execution paths, and operational effort across the SDLC.

This provides the following benefits:

- Predictable outcomes and cost discipline through task-aware execution, helping to ensure high-quality results on the first pass and reducing costly retries and rework.
- Dynamic routing selects the right model for the job, avoiding overuse of expensive models while preventing the hidden cost of low-quality outputs.
- Optimized execution paths reduce unnecessary compute and resource consumption across workflows.
- Enables predictable AI spend with pass-through pricing, centralized visibility, and policy-based controls (Bobalytics).

Shift-left without adding friction

IBM Bob shifts security and quality checks earlier in the lifecycle, helping teams prevent vulnerabilities and errors before they become costly production issues.

This provides the following benefits:

- Early detection in the IDE and pull request (PR) review
- Fewer incidents and less rework downstream
- Engineered trust through automated, auditable guardrails embedded at authoring time

Delivering the preceding benefits across SDLC requires different coverage. IBM Bob is designed to do just that:

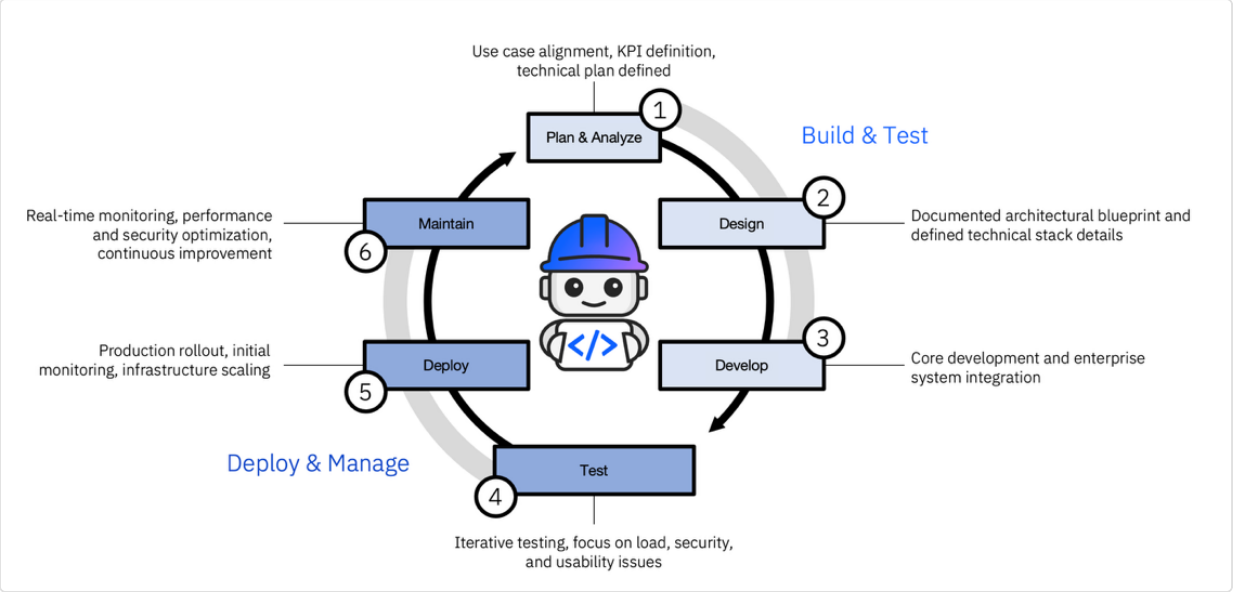


Figure: IBM Bob SDLC coverage cycle

This is made possible by combining multiple components. IBM Bob operates across different horizontal components, each with defined responsibilities and integration points:

- **Developer workstation:** VS Code, ABAP ADT or FS Extension, IBM Bob agent, LLM Engine, Context Studio MCP client, MCP Gateway client
- **MCP tool servers:** Multitude of MCP options ranging from ABAP ADT/FS, Context Studio, Jira, ServiceNow, SharePoint, SAP CALM, SAP BTP Administration, LeanIX, and many SAP open MCPs such as Fiori, UI5, CAP, and others
- **Network and auth boundary:** TLS 1.3, Bearer JWT, SAP ADT Auth (Basic/OAuth2), API Keys, Enterprise SSO
- **Generated artifacts:** SAP ABAP code, TSD/FSD docx, Jira/ServiceNow incidents, Context Nodes, Test reports, HTML artifacts, Transport Packages, and others

The following list describes role-based capabilities of IBM Bob:

- **Product Manager:** Requirement management, documentation generation of the features, feature feasibility analysis, API contract design, competitive analysis
- **Software Developer:** Component generation, API integration, form validation logic, performance optimization, data validation, error handling, debugging
- **QA Engineer:** Test case generation, unit test writing, integration test creation, E2E test automation, performance test scripts, test data generation, issue reproduction, regression test, and maintenance
- **SRE:** Incident response analysis, runbook generation, alerting rules creation, capacity planning, disaster recovery planning, performance tuning
- **DevOps Engineer:** Deployment pipeline generation, infrastructure as code, monitoring setup, log aggregation, security scan integration
- **Software Architect:** Technology recommendations, API gateway design, performance architecture, data architecture, competitive analysis
- **Data Engineer:** ETL pipeline creation, data validation rules, schema migration, data modeling, API data connectors, data documentation, data quality checks
- **Technical Writer:** API documentation, user guide creation, architecture documentation, release notes generation

Different use cases addressable

IBM Bob supports every phase of the development lifecycle, including managed scenarios, with no gaps and no handoffs. You can use IBM Bob across phases such as the following:

- **Planning and requirements:** To analyze needs, generate specs, shape architecture
- **Development:** Generate code across different languages and scenarios (for example, within SAP — classic, RAP, CAP)
- **Testing:** Write test scripts, validate, review, and execute testing scenarios while adjusting back as needed
- **Deployment:** Orchestrate scenarios across tools using MCPs
- **Maintenance and operations:** Monitor logs, optimize performance, fix root causes
- **Continuous improvement:** Reduce technical debt, improve workflows, enforce best practices

Key best practices to consider

Although later sections cover this in more detail in the context of SAP, consider the following key best practices at a high level:

- **Plan before you act:** Before asking a coding agent to act, plan and define your goals, scope the task, and structure how the transformation/actions should happen. This step helps you understand the scope of the task, create rules, and update the context.
- **Start small:** Don't try to migrate or generate the entire solution — begin with well-scoped MVPs, build well-defined skills and rules along with guardrails to provide enough context.
- **Design and maintain high-quality context:** A coding agent's output quality depends on the quality of its rules and of the context that the agent has about the project. Maintaining modular, specific, and readable rules along with context is an important part of project success.
- **Treat the coding agent as a co-developer:** Use interactive sessions when possible, review suggestions and perform pair programming to let the agent propose changes and modifications. As the human, you validate the proposed changes and fix and refine the strategy. Use version control to revert and review changes if needed.
- **Combine with automated testing:** Ask the coding agent to run tests or linters on changed files. This approach helps identify issues and iteratively fix them early on.
- **Human in the loop:** Humans actively review, validate, or approve AI-generated outputs before actions are executed, ensuring accuracy, compliance, and accountability for high-risk or critical decisions.

The next subsection explains how spec-driven development helps achieve optimized outcomes and how to adopt it compared with a pure vibe-coding-driven approach, mainly for production-grade solutions. See Vibe coding versus spec-driven development — What impact does it have with or without IBM Bob?

> **Note:** Vibe coding is still recommended, but mainly for pilot or MVP-based scenarios, where you can iterate actively with the client as the idea evolves and reduce the cycle time for initial discovery and preparation.

Vibe coding versus spec-driven development — What impact does it have with or without IBM Bob?

To see how the spec becomes the new code, consider how it differentiates between "Joy" and "Productivity":

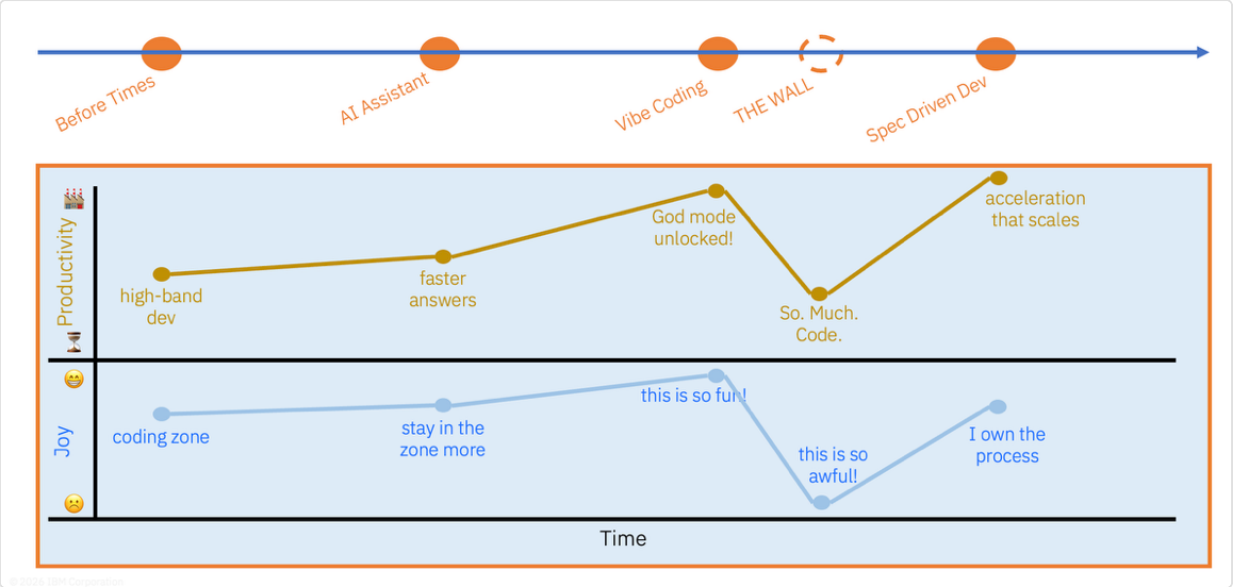


Figure: Vibe coding versus spec-driven development — Productivity and joy over time, Bill Higgins (IBM), Gabe Goodhart (IBM)

Have a clear point of view before you start using an IDE for code handling. This helps you avoid inconsistent outputs and overspending on tokens, which affects the ROI of the intended solution.

The following figure summarizes the criteria that differentiate typical prompt coding, vibe coding, and spec-driven development:

Criteria	Prompt Coding	Vibe Coding	Spec-Driven Development
Who Writes the Code?	You (with AI assistance on snippets)	The AI Agent (steered by your feedback)	The AI Agent (strictly bound by your document)
Best For...	Micro-tasks, debugging, or optimizing a single function	Rapid prototyping, MVPs, and quickly exploring ideas	Production-grade software and complex logic
The Primary Risk	Slow and repetitive for multi-file features	Code can turn messy and become hard to scale	Requires significant upfront time to write the specification
Human's Main Role	Active coder and syntax reviewer	Product director and visual tester	Technical architect and system designer

Figure: Prompt coding versus vibe coding versus spec-driven development comparison

So, what really is spec-driven development (SDD)?

Spec-driven development (SDD) is a software development methodology in which you write detailed specifications before implementing code. It emphasizes creating comprehensive, executable specifications that serve as both documentation and validation criteria for your code/software.

To break it down:

Specifications first

- What the software should do (behavior)
- Expected inputs and outputs
- Edge cases and error condition handling
- Performance requirements (non-functional requirements, or NFRs)

Executable specifications

- Can be validated automatically
- Can be used as acceptance criteria for tests
- Can be converted into living documentation

Iterative refinement

- Feedback from the stakeholders
- Technical feasibility analysis
- Discoveries during implementation

In summary, spec-driven development establishes repeatability through a well-defined constitution. The constitution flows into the different features and is then repeated across the project by all team members, driving central architecture and decisions. The following figure is an overview of the execution and adoption:

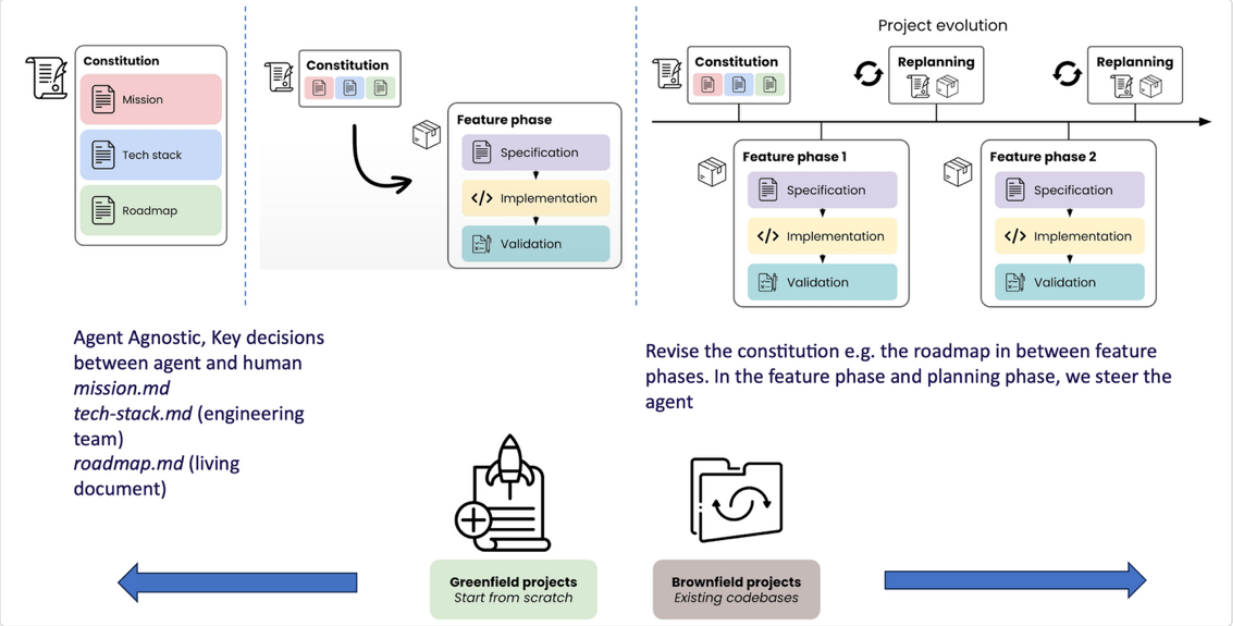


Figure: Spec-driven development — Constitution and feature phase execution model

This approach improves not only the efficiency and accuracy of the outcome but also repeatability and token optimization.

In the context of IBM Bob, tokens are expressed as Bobcoins.

How do Bobcoins relate to tokens?

Although you typically interact with LLMs in terms of tokens, IBM Bob conversations translate into "Bobcoins". At a high level, 1 Bobcoin equates to approximately USD 0.50, and there are different pricing models and T-shirt sizes available. Always refer to the current pricing model available on IBM Seismic — Bob pricing for more details.

Why use IBM Bob for SAP?

SAP environments present a distinct set of challenges that generic AI coding tools are not designed to address. This section examines those challenges in detail, explores how IBM Bob meets them, and positions IBM Bob relative to the Joule tools from SAP — including where the two complement rather than compete with each other.

Challenges

Different options available in the market today can handle SAP ABAP or SAP-related scenarios. However, having the context and being able to orchestrate end-to-end scenarios is the key requirement for an SDLC that is efficient and easy to adopt.

For example, the following figure summarizes the current SAP ABAP handling efficiency by the LLM itself:



Code Gen (after R5)	% of 1,800 runs passing after up to 5 feedback rounds — headline metric
Code Gen (1st attempt)	Same, but on the very first try with no feedback
Understanding (after R5)	Score on the separate ABAP understanding benchmark (180 tasks × 3 reps)
AUC R0–R5	Area under the cumulative-success curve; higher = improves faster with feedback
R0 Code Compiles (%)	% of first attempts where the code compiled & activated in SAP (regardless of test outcome)
pass@5	Probability that ≥1 of 5 randomly drawn runs passes (standard HumanEval metric)
Tasks Solved (≥1/10)	% of the 180 tasks where at least one run eventually succeeded
Tasks Solved (10/10)	% of the 180 tasks where every run succeeded (maximum consistency)

Filter Model

e.g. gpt, claude, qwen

Reset

Model	Code Gen (after R5)	Understanding (after R5)	Code Gen (1st attempt)	AUC R0–R5 (improves fast?)	R0 Code Compiles (%)	pass@5	Tasks Solved (≥1/10 runs)	Tasks Solved
gemini-3.1-pro-preview	88.33%	52.96%	44.33%	76.77%	46.56%	90.31%	91.11%	85.00%
gemini-3-flash-preview	84.44%	45.74%	10.44%	63.35%	11.56%	87.11%	87.22%	77.22%
GPT-5.3 Codex	84.39%	–	18.56%	62.45%	21.11%	88.27%	89.44%	76.67%
claude-opus-4-6	84.17%	85.37%	36.94%	69.43%	44.56%	89.39%	90.56%	76.11%
Claude Opus 4.5 (2025-11-01)	78.72%	61.11%	31.61%	65.03%	37.78%	80.90%	81.11%	76.11%
deepseek-reasoner	78.22%	75.56%	30.39%	62.12%	41.67%	87.28%	89.44%	54.44%
GPT-5 (2025-08-07)	77.11%	–	19.28%	56.60%	22.39%	84.70%	86.11%	59.44%
Claude Sonnet 4 (2025-05-14)	74.67%	–	24.11%	54.34%	29.06%	82.83%	84.44%	63.89%
GPT-5.2	64.00%	45.19%	16.33%	44.74%	20.00%	76.80%	78.89%	41.11%

Figure: SAP ABAP LLM benchmark results across frontier models

As the figure shows, LLMs reach an efficiency of at most 44.33% after the first attempt using the frontier models and become better only after approximately 3 to 4 iterations that maintain the context. This doesn't apply to all low- and medium-complexity scenarios and only comes into play with more domain-specific scenarios.

IBM Bob's Response

Consider the following key aspects:

- Context-Aware: knows codebase, conventions, project standards
- Full SDLC: requirements → production, no gaps
- Enterprise Security: code stays inside IBM/client network
- 200,000 token context windows (versus 64,000–128,000 for competitors)
- SAP-specialized: 50+ SAP MCP tools through ADT
- ICA Context Studio: project knowledge graph
- Governance: modes, skills, rules in git
- No custom RFC required: Uses standard ADT APIs (SAP and community-driven scenarios) Some known limitations apply to SAP standard ADT connectivity; later sections describe the details, and an enhanced premium package for SAP has been requested.
- IBM Granite models in the mix at the back end

Before later sections examine the use cases in depth, the following list shows different scenarios and how you can apply IBM Bob in the context of SAP:

- SAP ABAP:
  - Can generate code based on high-level requirements.
  - Structured code generation and human in the loop at each step give better control.
  - Code is generated following either the old or new ABAP syntax as needed.
  - No direct integration is mandatory if the client doesn't allow it, but integration options exist and are covered in Connecting IBM Bob to SAP.
- Integration:
  - Provides strong architectural insights on use cases for APIM, Edge Integration Cell, and API-first strategy
  - Accurate Groovy/XSLT generation for SAP CPI/Integration Suite
  - Complex orchestration-based integration scenarios
- UI5 and CAP:
  - Generates code following best practices for the SAPUI5 library and CAP
  - Prepares DevOps-ready unit test scripts and TS documentation
- MDG:
  - Generates MDG-specific data models into logical workshop agenda and generates the process flows
  - Able to write BRF+ rules, BADI, ABAP code
  - Reads WSDL or EDI structure to generate ABAP mapping

Although the preceding items are domain-specific samples, the following list shows some of the scenarios Bob can handle within the SAP context:

- Generate SAP ABAP code (Classic, RAP, CAP, UI5)
- Read, write, activate SAP objects live through ADT plug-ins For details about available plug-ins, what is possible, and what the known restrictions are as of today, see Connecting IBM Bob to SAP and Key known limitations as of 3Q 2026.
- Run ATC checks and ABAP unit tests
- Generate TSD/FSD documentation as docx in a template
- Query SAP tables (`SELECT` with context)
- Analyze ST22 dumps and traces
- Manage transport requests
- Connect to different systems beyond SAP, such as Jira, ServiceNow, and SharePoint
- Run AI-assisted debugging sessions
- Build architecture diagrams (Mermaid)

IBM Bob versus Joule

SAP Joule 4 Developers is a strong contender to IBM Bob and has native Eclipse-based integration for SAP landscapes driven by IBM proprietary SAP ABAP handling along with the additional context it brings forward. That said, it lacks client-specific knowledge and cannot generate documentation for legacy code or adopt project-specific best practices and naming conventions.

SAP Joule 4 Developers also lacks integration with other solutions that the client might have, especially with business context around the client's business processes and with non-SAP-based coding practices in the same context.

The following figure is a quick overview of how the tools fare:

Capability	IBM Bob	SAP Joule 4 Developers
Error Analysis		
Generate TSD/FSD for Legacy ABAP Program		
Leverage Client Knowledge	With Context Studio	
Generate ABAP Code		
Analyze ABAP Legacy Code		
Define Project Best Practices as Skills & Rules		
Easy to Integrate with Git & MCP		
Native Integration with SAP for code handling		
Performance Optimization		
RAP/Transactional App Generation		
Spec Driven Development		
MCP Integration with IBM Context Studio		
Intelligent SDLC		
Extend beyond SAP with same IDE		

Figure: IBM Bob versus SAP Joule 4 Developers capability comparison

Does IBM Bob replace SAP Joule 4 Developers and Joule 4 Consultants, or does it complement them?

Although the positioning of IBM Bob is clearly highlighted in the Why use IBM Bob for SAP? section, there are some scenarios where IBM Bob can coexist alongside SAP Joule 4 Developers and Joule 4 Consultants.

For instance, IBM Bob can be positioned as the central IDE solution for both SAP and non-SAP scenarios, while advanced RAP or SAP BAS-type coding scenarios can be handled with SAP Joule 4 Developers, which also provides broader coverage of different LLM models as your enterprise expands.

IBM Bob can also be used for knowledge search beyond just the SAP domain and for documentation/research purposes, but for SAP-specific scenarios and best practices, you can consider Joule 4 Consultants and extend as needed.

What are the challenges adopting IBM Bob for SAP?

In the following scenarios, IBM Bob adoption can pose a challenge:

- You cannot use the ABAP FS extension or community-driven MCPs across different scenarios and are awaiting standard SAP-released connectors.
- You are looking for domain-specific knowledge around advanced topics surrounding BRIM, CRM, and unique industry solutions — for which IBM Bob is not yet ready to handle the required functions.
- The client already has SAP Joule 4 Consultants and Joule 4 Developers, and it is a fully SAP-centric landscape.
- The client is looking for a complete on-premises deployment and cannot have any outside frontier models consumed through the product. (As of this writing, this is a limitation with IBM Bob. There are plans to address this in future releases.)

> **Note:** To address the preceding challenges, a premium package request for SAP on IBM Bob has been submitted. When the package becomes available, it should address these challenges and other requested features.

What to expect in the remaining sections

The remaining sections walk you through what the integration looks like with different systems, focusing mainly on SAP and its related ecosystem. They also provide input on configuration, workarounds (if any), and examples to help you be production-ready for your client.

The following figure is an overview of the integration landscape and how things evolve across the different phases of the project:

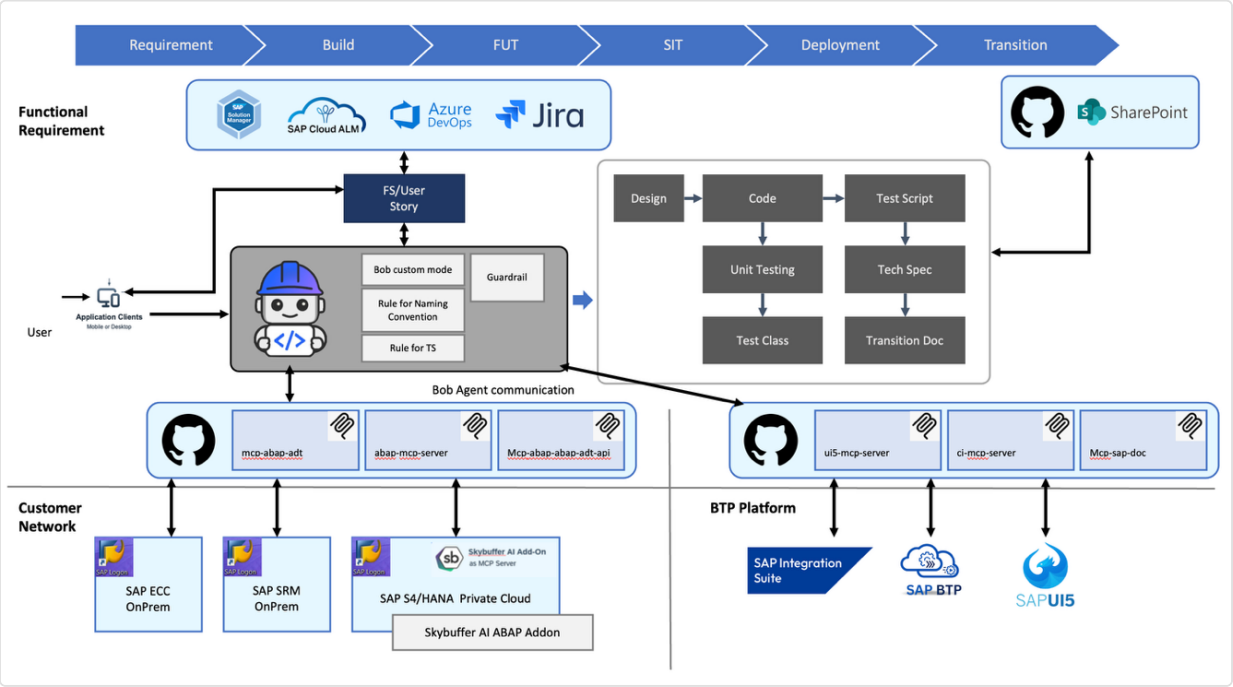


Figure: IBM Bob SAP integration landscape and SDLC pipeline

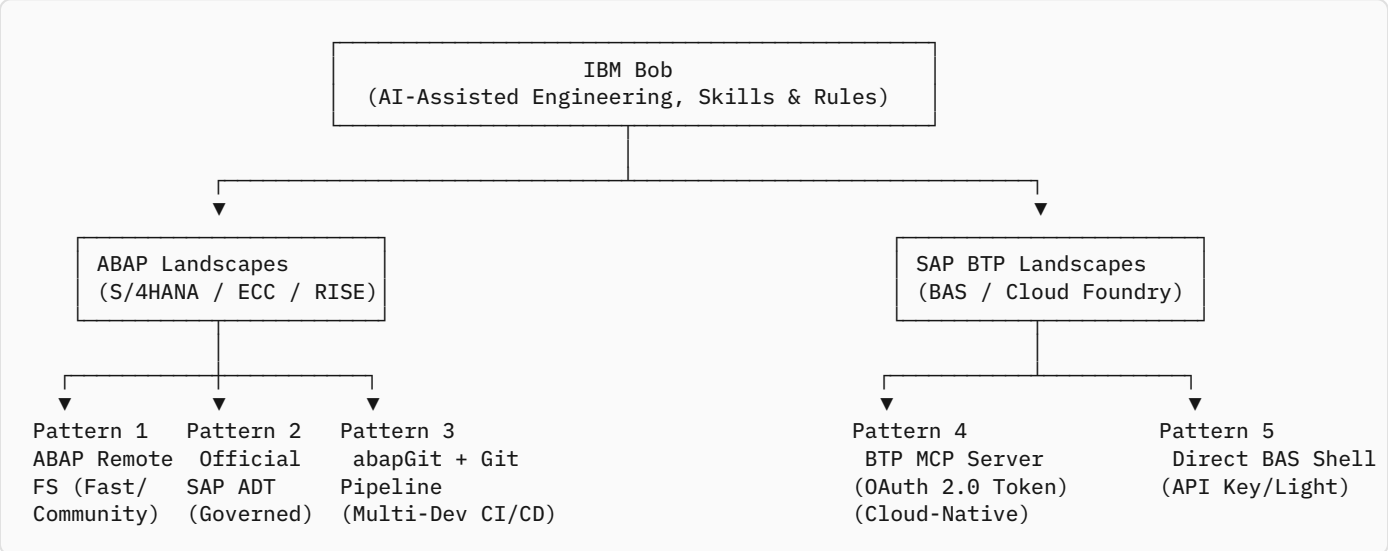
Footnotes

1. Gartner — Top 10 Strategic Technology Trends for 2025 ↩
2. Gartner — Top Strategic Trends in Software Engineering for 2025 and Beyond ↩ ↩[2]
3. Gartner — Market Guide for AI Governance Platforms ↩

Connecting IBM Bob to SAP

Enterprise AI is transforming software engineering, but AI deployed in a silo cannot deliver its full value. When an intelligent assistant operates without direct connectivity to an organization's operational backbone, developers are reduced to manual data gatherers—copying source code, table structures, and business context between systems. This disconnect creates a productivity paradox: time gained from AI acceleration is lost to context gathering, manual prompt preparation, and error-prone transport reconciliation.

This section establishes the definitive architectural blueprint and operating model for integrating IBM Bob directly into enterprise SAP landscapes spanning SAP S/4HANA (on-premises and RISE with SAP), SAP ECC, and SAP Business Technology Platform (SAP BTP).



What this section covers

This section covers the following topics:

- **The Back-End Integration Imperative**: The cost of disconnected AI silos and why real-time access to SAP transactional and development context is essential for meaningful AI-assisted development
- **Integration Patterns for SAP S/4HANA and ECC**: Comparative architecture, security tradeoffs, and use-case alignment for the community-driven ABAP Remote File System, the official SAP ADT extension for VS Code, and SAP ADT MCP server options
- **Integration Patterns for SAP BTP**: Deployment models for SAP Business Application Studio (BAS), contrasting an enterprise-grade Cloud Foundry MCP middleware with OAuth 2.0 against a lightweight direct shell integration

- **Governed Git-Driven Pipeline (abapGit + IBM Bob)**: Establishing a centralized, single-source-of-truth workflow that unites AI code generation, Git version control (branching, pull requests, code reviews), and native SAP CTS object activation
- **End-to-End Infrastructure and Security Configuration**: Practical configuration guides for installing abapGit (`ZABAPGIT`), configuring GitHub Personal Access Tokens (PAT), creating `SM59` RFC destinations over SSL, and defining proxy routing in `SICF`
- **Hands-on Development Walkthroughs**: Complete lifecycle execution for both greenfield ABAP program creation and brownfield legacy remediation, modernized for S/4HANA Clean Core standards

The back-end integration challenge

In today's evolving AI landscape, enterprises have access to a wide range of AI accelerators, tools, and utilities that can deliver measurable value in terms of accuracy, speed, efficiency, and productivity. However, one of the most significant challenges organizations face is the existence of AI in disconnected silos within the enterprise.

Most large enterprises operate within complex SAP ecosystems that support multiple business domains such as finance, billing, procurement, supply chain, and marketing. Each domain has its own operational processes enabled and managed through enterprise technology platforms. When issues occur within these environments, resolution activities are typically performed through changes or interventions at the SAP application layer, which acts as the operational backbone of the enterprise.

As organizations introduce Enterprise AI capabilities such as IBM Bob, an additional intelligent layer is effectively added on top of the existing technology architecture. This AI layer (that is, IBM Bob) can include capabilities such as issue-resolution agents, development assistants, reverse engineering support, operational intelligence, and automation services. However, the success of these AI capabilities depends heavily on seamless integration with the underlying SAP application landscape.

Without deep integration into the enterprise SAP ecosystem, AI solutions remain isolated and context deficient. In such scenarios, whenever a business or operational issue occurs, the AI system lacks direct access to the transactional, process, and operational context required to provide accurate and actionable responses. As a result, users are forced to manually gather relevant information and repeatedly construct detailed prompts to explain the business requirement or issue context.

This manual process introduces several operational challenges, including increased effort, higher risk of incomplete or inaccurate inputs, longer resolution times, and frequent switching between enterprise applications and AI tools. Ultimately, these inefficiencies reduce productivity, impact response accuracy, and limit the overall business value expected from Enterprise AI initiatives. The siloed AI paradox occurs if enterprises invest in IBM Bob but don't connect it to their operational backbone, SAP, and other applications. This creates the following problems:

- AI tools exist isolated but lack business context.
- They cannot understand real operational issues.
- They cannot access the data needed for intelligent decisions.
- Their recommendations become generic rather than enterprise-specific.

Therefore, tight integration between Enterprise AI systems and the SAP back-end landscape is essential. A fully connected architecture enables AI systems to access real-time enterprise context, business processes, application data, and operational workflows directly from the SAP ecosystem. Such integration transforms AI from an isolated productivity tool into an enterprise-grade intelligent platform capable of delivering scalable, contextual, and operationally relevant outcomes across the organization. This is why IBM Bob and SAP integration is non-negotiable.

The importance of using IBM Bob with SAP Integration

Seamless context access

Without integration to the SAP application layer, the following problems occur:

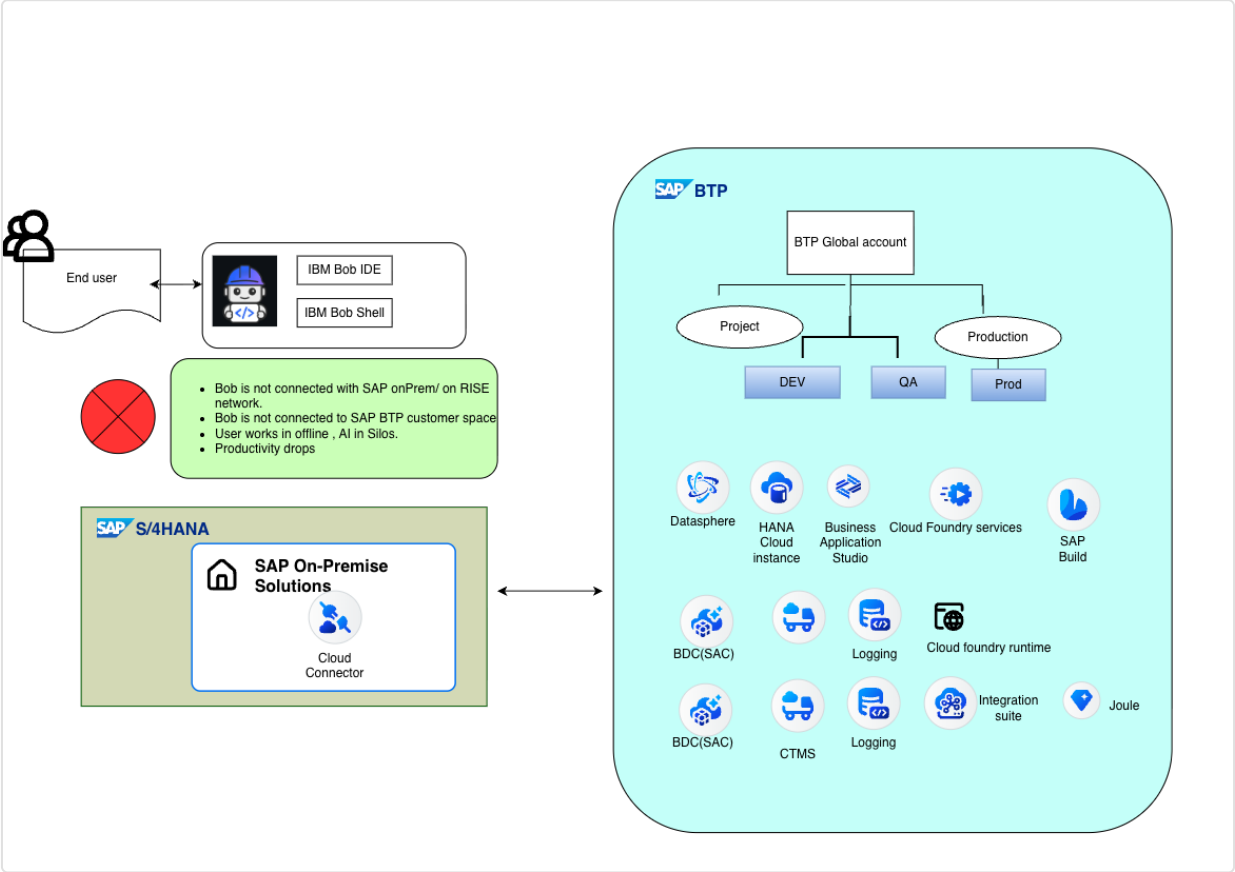
- Base code (ABAP, UI5, and so on) must be copied manually into the IBM Bob workspace to generate any output.
- IBM Bob operates on information users manually copy into prompts.
- Without integration, context is incomplete, fragmented, and inconsistent.
- Each interaction requires manual data gathering.
- Risk of miscommunication is exponentially higher.

The manual workaround cost

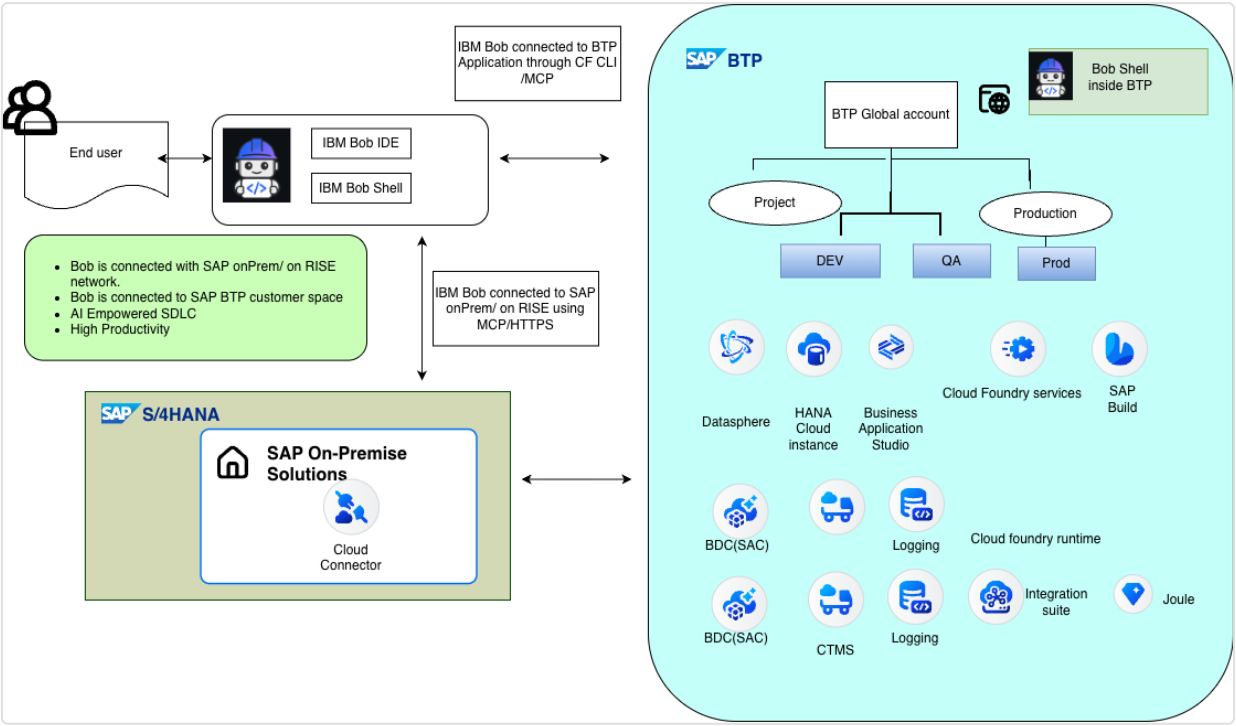
The current workaround (manual context injection) creates the following costs:

- Time loss: Switching between applications, constructing prompts
- Accuracy degradation: Incomplete or incorrect context transfer
- Productivity collapse: Negates the entire purpose of AI acceleration
- Risk amplification: Manual processes are error-prone

In the context of SAP, IBM Bob is envisioned to play a key role in ABAP and UI5 development. To facilitate this, establishing a connection between IBM Bob and ABAP Objects, as well as BTP Business Application Studio projects, is essential. This connectivity is a foundational element in building an Enterprise AI layer within the SAP ecosystem. A versatile connectivity layer incorporates a range of tools and methods to link IBM Bob (Enterprise AI) with SAP. Some of the potential tools and technologies used include MCP Tools, OAuth 2.0, REST API, JWT Token, SSH, ABAP FS MCP, VS Code ADT Extension, and IBM Bob Shell Service. The following sections explore these connectivity options in greater detail. See the following comparative diagram to gain insight into the importance of connecting the SAP back end with IBM Bob.



Disconnected architecture, IBM Bob with no SAP integration



Connected architecture, IBM Bob integrated with SAP on-premises and SAP BTP

Choosing an approach: Overview and comparison

Integrating IBM Bob with SAP landscapes

Organizations adopting AI-assisted software engineering increasingly need secure, practical ways to integrate external coding assistants with enterprise SAP environments. One such scenario involves integrating IBM Bob with customer SAP landscapes that span SAP S/4HANA or ECC systems, SAP RISE private environments, and SAP Business Technology Platform (SAP BTP) applications.

This section discusses deployment considerations, architectural constraints, and integration approaches for connecting IBM Bob to SAP environments hosted across customer-owned networks, hyperscaler infrastructure, and SAP-managed cloud services.

Deployment models for IBM Bob

IBM Bob can be deployed in multiple ways, depending on customer security, compliance, and operational requirements:

- **Stand-alone deployment** within the customer environment
- **Software-as-a-Service (SaaS)** consumption model
- **Private deployment** within a customer-controlled network or on-premises infrastructure
- **Hybrid deployment** combining SaaS and on-premises capabilities

These deployment options offer flexibility; however, the integration model must align with the hosting architecture of the SAP systems involved.

SAP landscape context

In many enterprise environments, SAP workloads are distributed across different hosting models:

- **SAP on-premises applications**, such as SAP S/4HANA or SAP ECC, can run in a customer private network or in a RISE with SAP environment hosted on a customer-designated hyperscaler such as Amazon Web Services (AWS) or Microsoft Azure.
- **SAP BTP applications** are typically hosted within SAP-managed secure networks, running on SAP-operated hyperscaler infrastructure.

As a result, integrating an external AI coding assistant such as IBM Bob into this ecosystem is not a simple point-to-point connection. It requires careful consideration of network boundaries, authentication mechanisms, security controls, and development tool integration patterns.

At a high level, the integration can be divided into two distinct areas:

1. Integration of IBM Bob with SAP S/4HANA or ECC environments (on-premises or RISE)
2. Integration of IBM Bob with SAP BTP applications

Integrating IBM Bob with SAP S/4HANA or ECC environments

The first integration scenario focuses on enabling IBM Bob to interact with SAP ABAP-based environments, including access to the following development artifacts:

- ABAP Programs
- Data Dictionary (DDIC) objects
- Classes and methods
- Enhancements and modifications

- Message objects, and so on

The objective is to allow developers to use IBM Bob to analyze, generate, and modify SAP development objects in a seamless and secure manner, enabling AI-assisted development directly within the SAP ABAP ecosystem.

Key challenge: Cross-network connectivity

The primary challenge is architectural: IBM Bob might reside outside the customer's internal SAP network (either in IBM-managed cloud infrastructure or as a SaaS offering), while the SAP S/4HANA or ECC systems remain protected within a customer-controlled private network boundary. This means the integration must securely bridge IBM Bob and an internal enterprise system without compromising security, governance, or data residency requirements.

Enabling mechanism: SAP VS Code extension

A relevant and practical enabler for this integration is the availability of a Visual Studio Code extension, provided by SAP, that facilitates seamless interaction with SAP development environments. This extension serves as the integration point between the developer's local workspace, IBM Bob, and SAP ABAP systems, enabling remote development object manipulation.

Integration approaches for part 1

To support this integration model, four alternative approaches have been proposed for integrating IBM Bob with SAP S/4HANA or ECC environments. Although the specific choice depends on customer architecture, security policy, network topology, and operational requirements, the general objective remains consistent: provide controlled, authenticated access to SAP development objects while preserving network isolation, compliance requirements, and audit trails.

Approach 1

Install "ABAP remote filesystem ADT" (community-driven) Extension in IBM Bob and integrate SAP back end with IBM Bob

In this approach, IBM Bob uses the extension to connect to SAP ECC and SAP S/4HANA back ends. Bob supports development activities, issue analysis, and resolution across the SAP landscape.

Approach 2

Install "ABAP Development Tools for VS Code" (SAP released) Extension in IBM Bob and integrate SAP back end with IBM Bob

In this approach, IBM Bob uses the extension to connect to SAP systems on-premises and in the cloud. Bob supports development activities, issue analysis, and resolution across the SAP landscape.

Approach 3

Configure SAP ADT MCP (community-driven) Server in IBM Bob

In this approach, IBM Bob interacts with the ABAP system through an SAP ADT MCP server. This enables Bob to communicate with ABAP systems, retrieve source code, inspect table structures, analyze development objects, and support intelligent development and troubleshooting workflows.

Approach 4

Configure SAP ABAPGit (community-driven) in SAP and connect with IBM Bob and IBM GitHub

In this approach, IBM Bob interacts with the ABAP system through SAP ABAPGit and any Online repository (IBM GitHub). This enables Bob to communicate with ABAP systems, pull and push source code changes, inspect table structures, analyze development objects, and support intelligent development and troubleshooting workflows.

The selection among these four approaches should be based on the following criteria:

- Network connectivity model (VPN, private network gateway, secure tunnel, and so on)
- Authentication and identity federation requirements

- Compliance and data governance frameworks
- Operational and support considerations
- Developer experience and tool familiarity

Integrating IBM Bob with SAP BTP applications

The second integration scenario concerns SAP BTP applications, including development environments such as SAP Business Application Studio (BAS) and UI5-based applications. Since SAP BTP is hosted within SAP-managed cloud infrastructure, this integration typically faces fewer network boundary challenges than the previous section; however, security and authentication considerations remain paramount.

In this context, two primary integration approaches can be considered.

Approach 1: Integration through SAP BTP using Cloud Foundry CLI and an MCP server

In this approach, IBM Bob is integrated with SAP BTP through the following actions:

1. Using the Cloud Foundry command-line interface (CLI) to provision and manage cloud-native services
2. Creating an MCP (Model Context Protocol) server that acts as a middleware layer
3. Establishing connectivity tools and secure communication channels
4. Connecting IBM Bob with SAP Business Application Studio using OAuth 2.0 token-based authentication

Architecture

IBM Bob → MCP Server → OAuth 2.0 Token → SAP Business Application Studio

This approach offers the following benefits:

- **Enterprise-grade security** through OAuth 2.0 token-based authentication
- **Cloud-native architecture** fully aligned with the Cloud Foundry runtime of SAP BTP
- **Scalability and resilience** through managed cloud services
- **Governance and auditability** of all interactions between IBM Bob and BTP applications
- **Integration with SAP's identity and access management** infrastructure

The MCP server acts as a controlled gateway, enforcing policy, managing authentication, and potentially enriching the interaction with additional business logic or compliance controls.

Approach 2: Direct shell integration with API key authentication (lighter-weight option)

A second option is to install the IBM Bob shell directly within the SAP Business Application Studio terminal and authenticate using an IBM Bob API key.

Architecture

IBM Bob Shell → BAS Terminal → IBM Bob API Key Authentication → AI Empowered BAS

This model provides the following benefits:

- Simpler and faster deployment with minimal infrastructure dependencies
- Developer-friendly experience through direct terminal access
- Reduced operational overhead compared to MCP server management
- Suitable for development and non-production environments

Architectural considerations and best practices

When evaluating either integration path, organizations should systematically assess the following dimensions:

- Network Topology and Connectivity:
  - **For Part 1 (On-Premises SAP)**: Evaluate whether IBM Bob can securely reach on-premises SAP systems through secure tunnels (VPN, AWS PrivateLink, Azure ExpressRoute), private network gateways, or other network isolation mechanisms.
  - **For Part 2 (SAP BTP)**: Use SAP-managed network infrastructure and cloud-native connectivity patterns.
- Authentication and Authorization:
  - **Token-based authentication** (OAuth 2.0 recommended for production)
  - **API key management** and rotation policies
  - **Role-based access control (RBAC)** aligned with SAP identity frameworks
  - **Federation** with enterprise identity providers (Azure AD, SAP BTP Identity Authentication Service)
- Data Residency and Compliance:
  - **Code and metadata handling** across network boundaries
  - **Regulatory compliance** (GDPR, SOX, industry-specific standards)
  - **Data encryption** in transit and at rest
  - **Audit and logging** requirements for AI-assisted development activities
- Operational Governance and Lifecycle Management:
  - **Auditability** of all IBM Bob interactions with SAP systems
  - **Monitoring and alerting** for unusual activities
  - **Policy enforcement** for code generation, modification, and quality standards
  - **Version management** of IBM Bob and integration components
  - **Support model** and vendor responsibility matrix

Integrating IBM Bob with SAP landscapes requires a differentiated, architecture-aware approach based on where SAP workloads are deployed and how network trust boundaries are defined.

For SAP S/4HANA or ECC systems in on-premises or RISE environments, the integration challenge centers on securely bridging IBM Bob with SAP systems protected within customer-controlled private networks. This requires careful attention to network connectivity, authentication federation, and compliance frameworks.

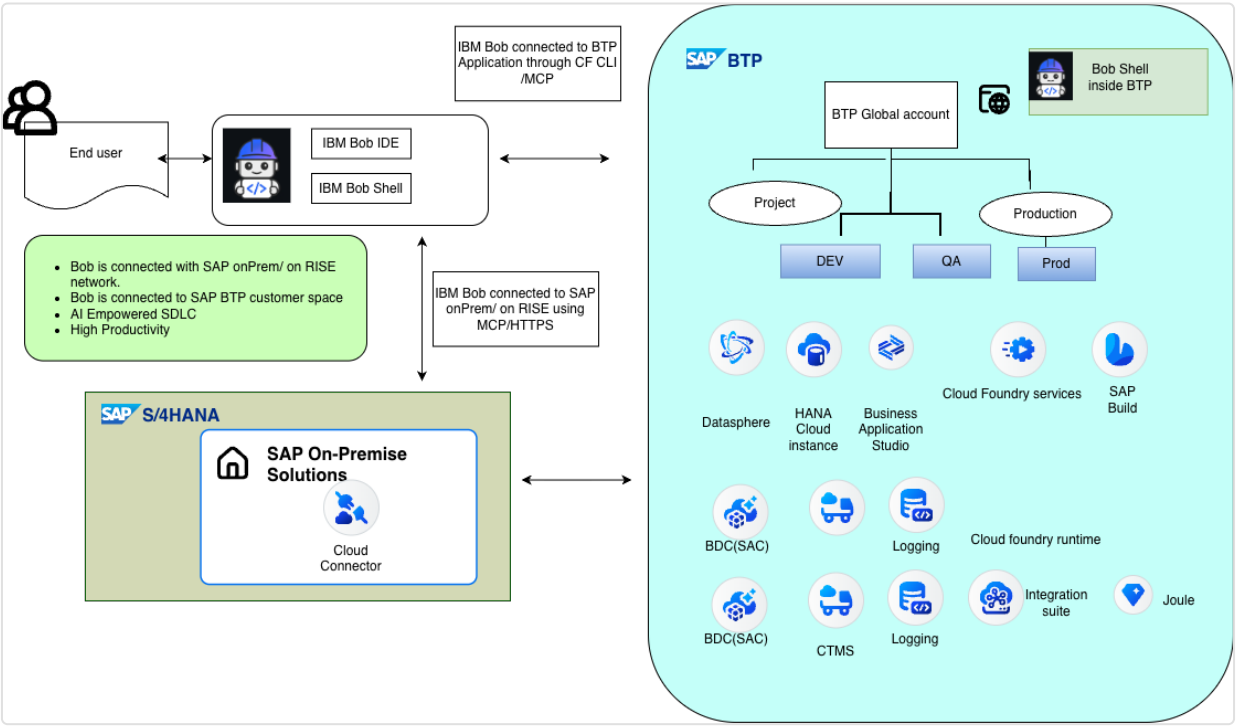
For SAP BTP applications, integration is comparatively more straightforward and can be achieved through two models:

- **Approach 1 (Cloud Foundry + MCP + OAuth 2.0)** for robust, enterprise-grade production deployments
- **Approach 2 (Shell + API Key)** for lighter-weight development and POC scenarios

Together, these approaches establish a comprehensive foundation for bringing AI-assisted development capabilities into heterogeneous SAP environments while maintaining the security, governance, compliance, and operational flexibility required by enterprise customers.

The success of IBM Bob integration ultimately depends on aligning the deployment model with organizational architecture, security policies, development practices, and business requirements—ensuring that AI-assisted development becomes an accelerator rather than a source of operational or compliance risk.

The enterprise architecture using IBM Bob looks like the following. This overview provides context for the detailed discussion of each topic.



Enterprise architecture overview, IBM Bob connected to SAP and SAP BTP

ABAP remote FS extension for IBM Bob

Imagine a mid-sized manufacturing company that has been running SAP ECC for over a decade. Their ABAP developers are skilled, their code base is substantial, and their business teams are asking for faster delivery. Someone in IT leadership has heard about AI-assisted development and wants to know: how quickly can we get IBM Bob working with our SAP code?

For organizations like this, the answer lies with the ABAP Remote File System Extension — a community-driven integration, built and maintained by the open-source community, not by SAP itself. That distinction shapes everything about how it works, what it offers, and what organizations need to bring to the table themselves.

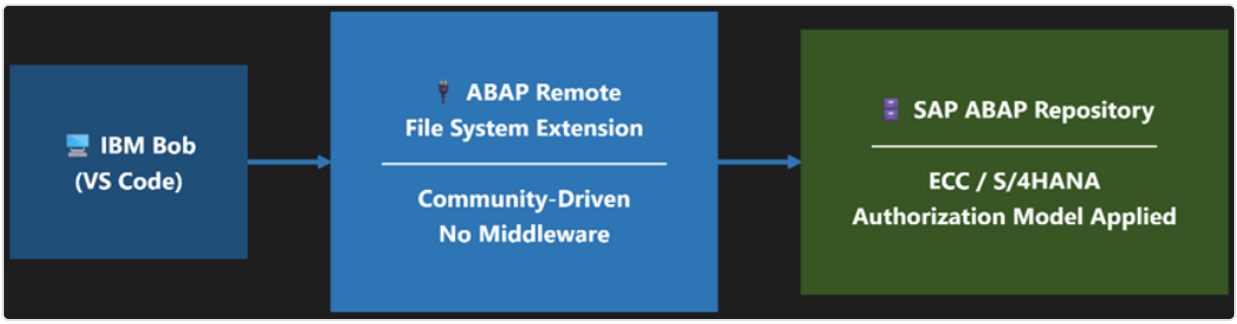
Removing the obstacles

Not every organization needs — or is ready for — a fully architected enterprise integration. Sometimes the most valuable thing a tool can do is get out of the way and let people start working.

The ABAP Remote File System Extension, maintained by the community, does exactly that. It creates a direct bridge between SAP's ABAP code layer and IBM Bob, without requiring a dedicated integration server, a custom API layer, or weeks of middleware configuration. There are fewer components to install, fewer moving parts to manage, and fewer reasons to delay.

For the manufacturing company's developers, that means IBM Bob can be connected to their SAP system within minutes of setup. Code that has been sitting in the system for fifteen years — programs that newer team members struggle to understand and senior developers barely remember writing — suddenly becomes accessible. IBM Bob can read through that code, explain what it does, identify areas of concern, and help developers work with confidence in an unfamiliar code base.

That is not a small thing. In many SAP landscapes, the biggest barrier to faster delivery is not technical skill — it is the sheer weight of undocumented legacy code that nobody feels comfortable touching. The Remote File System Extension, combined with IBM Bob, begins to lift that weight.



Direct connection from IBM Bob to the SAP ABAP repository with no middleware

Security without reinventing the wheel

A natural question arises when introducing any new tool into a SAP landscape: what does it do to our security model?

The honest answer here is: very little — and that is a good thing.

The ABAP Remote File System Extension continues to operate within SAP's existing authorization framework. Developers can only access the programs, objects, and data they are already permitted to see. No new access paths are created. No parallel permission system is introduced. The authorization model that the organization has already defined and validated continues to govern what IBM Bob can reach.

This means the security conversation is shorter than many organizations expect. There is no need to design a new role concept, validate a new authentication mechanism, or explain to the security team why a third-party tool is bypassing SAP's controls. It is not bypassing them — it is working through them.

Understanding the tradeoff honestly

Every architectural choice involves a tradeoff, and this one is worth being direct about.

The ABAP Remote File System Extension earns its speed and simplicity by depending less on built-in governance mechanisms and more on the organization's own processes and disciplines. There is no dedicated enterprise governance framework baked into the approach. Transport discipline, peer code review, change management procedures — these remain the organization's responsibility to enforce, just as they were before IBM Bob arrived.

For the manufacturing company in our example, this is likely not a problem. Their development practices are established. Their team knows how changes move through the landscape. Adding IBM Bob as a productivity tool does not fundamentally alter the way they manage development risk — it gives their developers a faster, smarter way to work within the practices they already follow.

For an organization with less mature development processes, the picture looks different. The Remote File System Extension will still work, but the governance gap will be more visible. In that case, the right response is not to avoid the tool — it is to invest in the processes alongside it.

Where this approach fits best

The ABAP Remote File System Extension is not a universal answer, but it is the right answer for a clearly defined set of situations:

- Incident Analysis: Developers can access affected code immediately for fast diagnosis and resolution. Speed of access translates directly to speed of resolution.
- Application Management Services (AMS): Day-to-day support teams benefit most when responsiveness matters more than architectural sophistication.

- Small Enhancements: Minor changes delivered without the overhead of a complex tool landscape, keeping delivery lean in stable SAP environments.
- Legacy ECC Environments: Systems that do not require migration can be maintained and analyzed more effectively, extending the productive life of existing investments.

> **Note:** The ABAP Remote File System Extension offers the fastest path to connect SAP code with IBM Bob, delivering rapid productivity gains while placing greater responsibility for governance, security, and compliance on the organization. Best suited to teams with established development disciplines who value speed of access over architectural governance.

ABAP development tools extension for Visual Studio Code

Imagine a large financial institution running hundreds of custom ABAP programs across multiple SAP landscapes. Their development teams are experienced, their governance processes are mature, and their compliance obligations are strict. When they evaluate how to bring IBM Bob into their SAP environment, the question is not just whether it can connect; it is whether we can trust it, audit it, and stand behind it when the regulator asks.

For organizations like this, the SAP ADT Extension is the natural answer — and what sets it apart from the other paths before it is a single, important fact: this is an officially supported SAP product. It is released and maintained by SAP, aligned with SAP's product strategy and roadmap, and backed by SAP's commitment to its long-term evolution. Organizations are not relying on community goodwill or volunteer maintenance cycles. They are building on the same foundation SAP uses to develop its own tools.

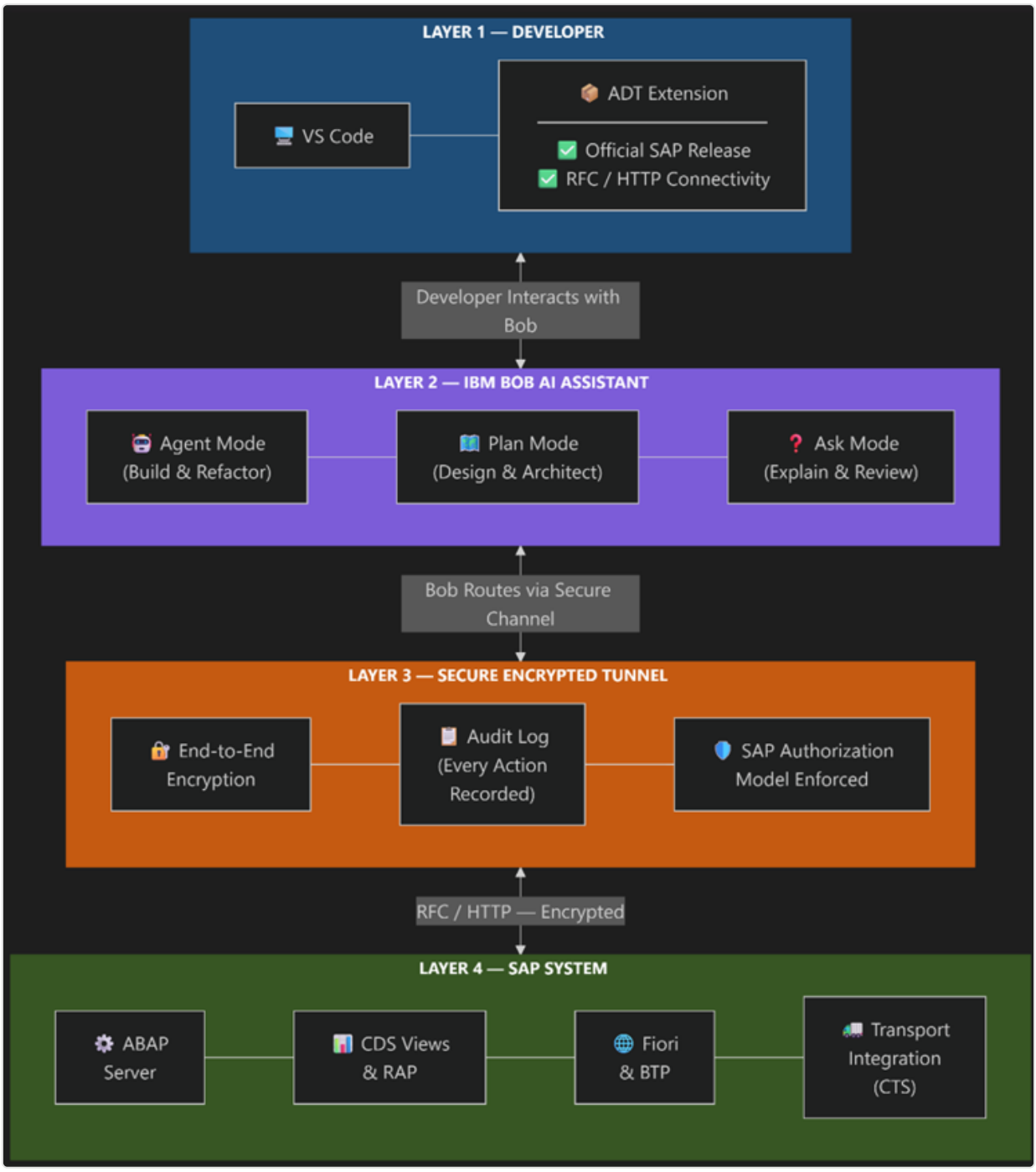
Built on familiar ground

SAP's ABAP Development Tools — better known as ADT — did not arrive as an afterthought. It is SAP's own strategic IDE, built on Eclipse and available as a VS Code extension, and represents the direction SAP has chosen for ABAP development going forward. It supports the full spectrum of SAP development: from classical ABAP on traditional systems to ABAP Cloud, the RESTful Application Programming Model (RAP), and development on SAP BTP. Debugging, testing, transport integration, and cloud-native development paradigms are all part of what the extension brings to the table.

When IBM Bob connects through the ADT Extension, it is not working around SAP's architecture — it is working within it. That distinction matters enormously to enterprise organizations.

Connectivity is supported through RFC for on-premises and SAP S/4HANA Cloud Private Edition, and through HTTP for SAP BTP ABAP environment and SAP S/4HANA Cloud Public Edition. The extension applies to the following standard releases:

- On-premises/Private Cloud: SAP NetWeaver 7.3 EHP1 SP04 or higher, SAP S/4HANA 1709 FPS00 or higher, SAP S/4HANA Cloud Private Edition 1709 FPS00 or higher
- Public Cloud: SAP BTP ABAP environment 2608, SAP S/4HANA Cloud Public Edition 2608



Four-layer architecture of IBM Bob connecting through the SAP ADT Extension

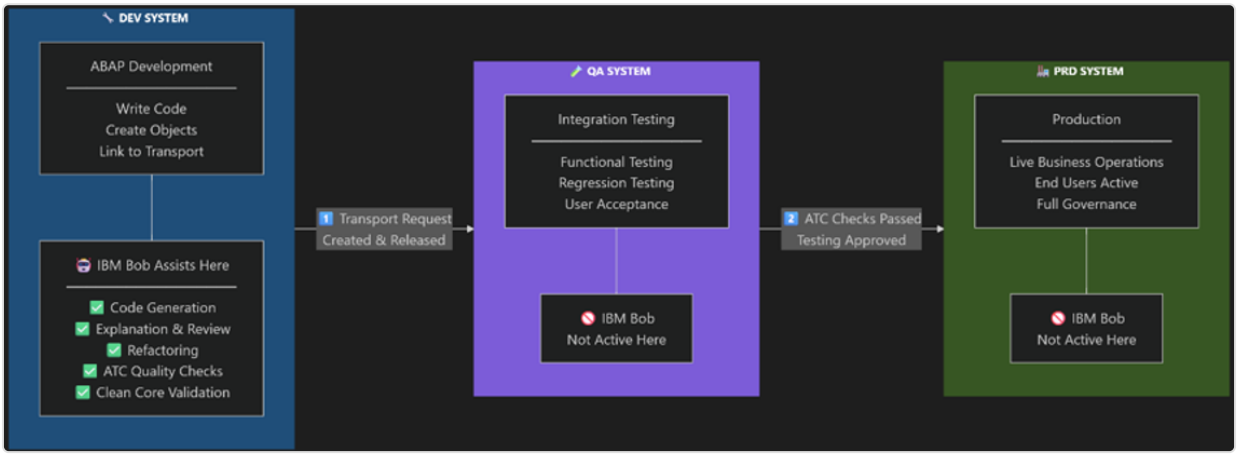
Governance that comes standard

Consider what happens when a developer makes a change in a large SAP landscape. That change does not only appear in production. It moves through a structured lifecycle — development, quality assurance, production — carried by SAP's Change and Transport System. Every modification is tracked, every transport is logged, and every promotion through the landscape leaves a clear audit trail.

The ADT Extension preserves all of that. Native transport integration means that development changes made with the assistance of IBM Bob continue to follow the same standard SAP lifecycle they always have. Nothing is bypassed. Nothing is introduced outside the established process.

At the same time, because the extension operates entirely within SAP's authorization framework, access controls do not need to be redesigned or supplemented. The roles and authorization objects that organizations have spent years defining continue to govern what developers can see and do. For compliance teams in regulated industries — banking, utilities, pharmaceuticals — this is not a minor detail. It is often the deciding factor.

The overall security posture is strong — not because new security mechanisms have been introduced, but because the extension relies on familiar SAP security concepts that organizations already understand, validate, and trust. There is no need to review alternative authentication schemes or justify third-party middleware to a security team.



SAP CTS transport lifecycle with IBM Bob scoped to the DEV system

A powerful companion, not a replacement

The tool meaningfully enhances developer productivity — helping with code explanation, issue identification, routine analysis, and the kind of repetitive work that consumes developer time without adding much value. For large SAP delivery teams, those gains accumulate quickly.

What it does not do — at least not yet — is replace the developer entirely. IBM Bob, in this context, is a powerful coding companion. The developer remains responsible for decisions, the organization remains in control of its governance processes, and the accountability that enterprise SAP development demands is fully preserved.

For the financial institution imagined at the start of this section, that is precisely the balance they were looking for.

Where this approach fits best

The SAP ADT Extension supports the widest range of SAP development scenarios of any integration approach:

- Modern ABAP Development: Classical ABAP, OO ABAP, ABAP Cloud, and CDS Views in a single environment
- RESTful Application Development (RAP): CDS Data Models, Behavior Definitions, Service Definitions, OData Services. Ideal for S/4HANA Public Cloud and Clean Core initiatives
- ABAP Cloud Development: Released APIs only, Clean Core compliant extensions, side-by-side extensibility
- AI-Assisted Code Generation: Generate ABAP code, CDS objects, RAP artifacts, boilerplate classes
- Code Explanation and Knowledge Transfer: Reduces dependency on specialist subject matter experts
- Code Refactoring and Modernization: Procedural to OO ABAP, performance optimization, ECC to S/4HANA transformation support
- ATC-Based Quality Governance: ATC checks, Clean Core compliance, performance and security checks, S/4HANA readiness
- Transport Management: Object activation, package management, change tracking with full enterprise auditability

- Clean Core Assessment and Remediation: Reduce modifications, adopt released APIs, move to compliant frameworks
- Hybrid Development: ABAP, CAP, Node.js, UI5/Fiori, BTP in one IDE. Ideal for full-stack SAP BTP developers
- Legacy ECC Modernization: Code remediation, custom code analysis, S/4HANA readiness, brownfield S/4HANA conversions
- Developer Productivity: Autocompletion, quick navigation, multi-object activation, search across repositories

SAP ADT MCP server: The intelligent gateway

Organizations that operate SAP systems often need to understand ABAP application behavior quickly and consistently. The required evidence is commonly distributed across source code, development packages, dictionary definitions, object metadata, and technical relationships. Finding and interpreting that evidence can be time-consuming, particularly when teams are supporting established systems, investigating incidents, assessing change impact, or onboarding engineers.

An SAP ABAP Development Tools (ADT) Model Context Protocol (MCP) server can provide IBM Bob with a structured way to request selected development-time information through configured ADT services.

ADT is SAP's development interface for working with ABAP repository artifacts. MCP is a protocol that lets an AI-enabled client use defined tools supplied by an external service. In this configuration, IBM Bob can request information that the MCP server exposes and that the connected SAP system authorizes.

This capability should be understood as *controlled remote inspection*, not unrestricted control of an SAP system. The available operations, the scope of the returned information, and the ability to access individual objects depend on the ADT services that are configured, the MCP server implementation, network connectivity, and the permissions of the authenticated SAP user. A prudent implementation begins with the minimum necessary services and read-oriented access, then validates the results and governance model before expanding use.

Business challenge

ABAP applications can have long operational lives and broad dependencies. A single business behavior might involve classes, function modules, reports, interfaces, database tables, Core Data Services (CDS) artifacts, package assignments, and enhancement points. Relevant technical knowledge can be spread across repository objects and held informally by experienced personnel.

Traditional investigation methods often require a developer or support analyst to navigate several development tools, search by name, open related objects, and manually assemble a picture of the solution. This approach is appropriate for detailed development work, but it can slow early-stage questions such as:

- Which package owns a reported object?
- What fields are defined in a table or structure?
- Which source object contains a referenced routine or class?
- What repository relationships should be reviewed before a change?
- Which technical artifacts are relevant to an incident description?

The challenge is not only locating data. Teams need a governed method for retrieving technical context, explaining what it means, and keeping human engineering judgment in the decision loop. They also need to avoid treating an assistant's interpretation as a substitute for design review, testing, or authorization controls.

Solution overview

The solution consists of an SAP ADT MCP server configured as a connectivity endpoint for IBM Bob. The MCP server acts as an intermediary: it presents a defined set of tools to IBM Bob and translates permitted requests into calls to

configured SAP ADT services. Rather than giving a client broad, undefined access to an SAP landscape, this model limits interactions to the service capabilities and credentials intentionally made available.

At a high level, an authorized user asks IBM Bob a technical question in natural language. IBM Bob determines whether an available MCP tool can help answer the question, submits a structured request, and receives a structured result. IBM Bob can then help organize the result into an explanation, a code-comprehension aid, an impact-assessment starting point, or a troubleshooting hypothesis. The retrieved material remains technical evidence that should be verified by qualified SAP personnel when decisions or changes are involved.

A typical configuration establishes the following boundaries:

1. **An SAP ADT service endpoint** exposes the repository information that the SAP system and configuration allow.
2. **An MCP server** defines the tools, inputs, outputs, and connection handling that IBM Bob can use.
3. **IBM Bob** uses those declared tools to request relevant information and present it in context.
4. **Authentication and authorization controls** determine which calls succeed and which repository objects are visible.
5. **Operational governance** determines who may use the integration, what data may be retrieved, and how activity is monitored.

The exact tools and information available vary by MCP server version, its configuration, the target SAP system, and the user authorization model. The integration should therefore be documented and tested in the specific environment rather than assumed to provide any operation.

Understanding MCP-based connectivity

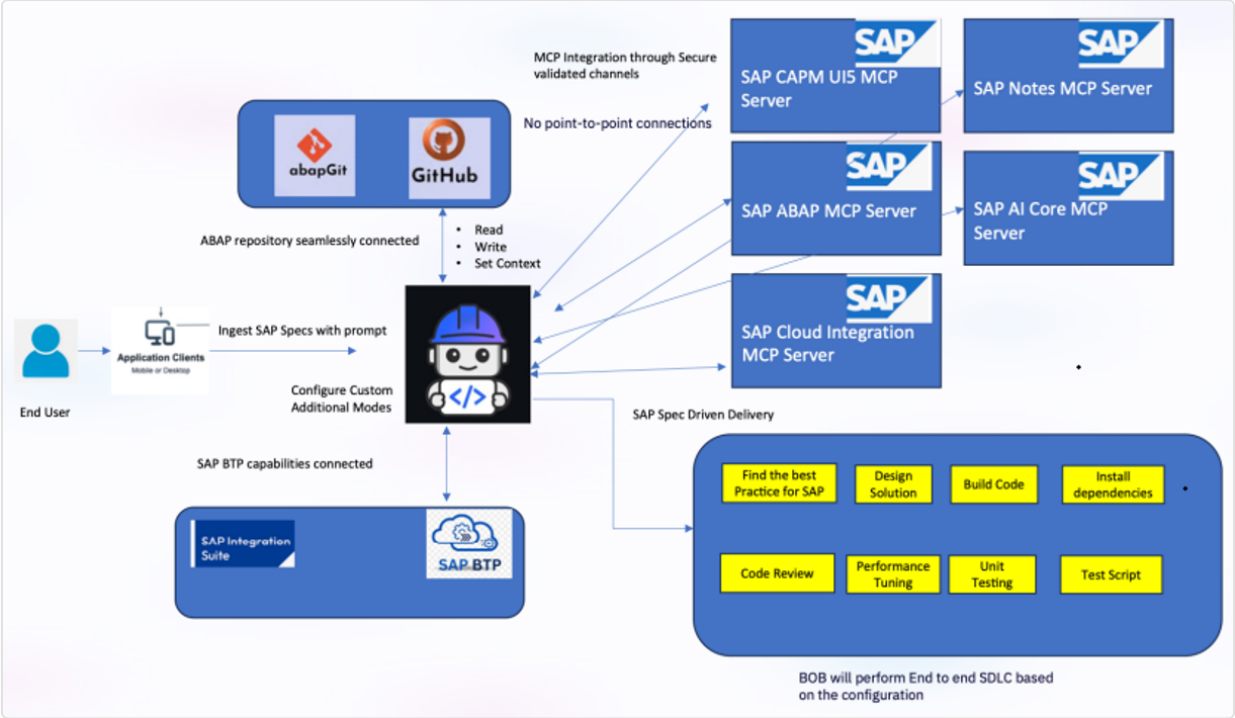
The Model Context Protocol (MCP) is a protocol for connecting an AI-enabled client to external tools and information sources through a defined interface. In this pattern, an MCP client does not need direct knowledge of every underlying system protocol. Instead, it discovers and invokes tools that an MCP server explicitly offers.

For SAP ADT connectivity, the MCP server is the controlled integration layer. A tool may represent an operation such as retrieving a source object, searching a package, obtaining a table definition, or returning metadata for a repository artifact. The tool contract specifies the request parameters and result format. This structure is important because it makes the intended interaction explicit and enables validation, logging, and policy enforcement at appropriate points in the architecture.

MCP-based connectivity does not change SAP authorization principles. The MCP server can only retrieve information when the underlying ADT service is available and the authenticated SAP identity is allowed to access it. Similarly, an MCP server cannot safely be presumed to offer write, activation, transport, or administrative operations merely because it connects to SAP. Such capabilities, if implemented at all, require separate design, explicit authorization, and heightened governance.

The practical value of the protocol is therefore not unrestricted access. It is a consistent mechanism for making selected, governed technical inspection functions available to IBM Bob.

Architecture and communication flow



IBM Bob connected to SAP through multiple MCP servers

A conceptual architecture contains four primary elements: the IBM Bob client experience, the MCP server, the SAP ADT services, and the SAP authorization and repository layers.

The communication flow proceeds as follows:

1. A user asks IBM Bob a question, for example, “Show the definition of the structure used by this interface.”
2. IBM Bob identifies an available ADT MCP tool that is relevant to the question. It prepares a structured request using the tool's supported inputs, such as an object name or package identifier.
3. The MCP server receives the request and applies its configured behavior. Depending on the implementation, this can include input validation, target selection, connection handling, request logging, and error translation.
4. The MCP server calls the configured SAP ADT service. The SAP system authenticates the request and evaluates authorization for the requested operation and object.
5. If access is granted and the service supports the request, SAP returns structured repository information to the MCP server.
6. The MCP server returns the result to IBM Bob. IBM Bob uses the returned evidence to answer the question, identify follow-up artifacts, or explain relationships in terms useful to the user.

This flow separates three responsibilities. IBM Bob helps interpret questions and results. The MCP server provides a constrained tool boundary. SAP remains the source system that evaluates service availability and authorization. The separation is useful for control: an answer generated by IBM Bob should be traceable to the tool result, and a successful retrieval should remain subject to SAP and integration-layer controls.

Network design also matters. Organizations commonly restrict the communication path to approved hosts and ports, use encrypted transport where supported, and avoid placing development-service endpoints on untrusted networks. Connection details, certificates, and tokens should be handled as sensitive configuration data, not embedded in prompts or documentation.

Information accessible through MCP

When the configured ADT services and permissions allow it, the integration can provide structured access to development-time information. The following categories are common:

- **ABAP source code:** The text and organization of authorized repository objects, such as classes, interfaces, programs, includes, and function modules. Source retrieval can support explanation and review, but it does not establish that the source is current, activated, or behaviorally correct without additional validation.
- **Package and object data:** Package assignment, object identity, object type, and related repository organization details. A package is an ABAP repository container used to group development objects and support transport and lifecycle management.
- **Dictionary definitions and structures:** Technical definitions of database tables, views, data elements, domains, structures, and fields, subject to the services exposed. The ABAP Dictionary is the SAP repository for these reusable data definitions. Understanding a field's type, length, key status, or semantic reference can clarify interface and persistence questions.
- **Development metadata:** Attributes that help describe an artifact, such as its name, type, description, package, status indicators, or other available repository properties. The specific metadata returned depends on the ADT endpoint and tool contract.
- **Technical relationships:** References and associations among repository artifacts where the available services can provide them. These relationships can help identify candidates for review, but they might not represent every runtime dependency, dynamic call, configuration dependency, or enhancement path.
- **Repository details:** Additional object-level information exposed by configured services, such as locations, hierarchy information, or supported navigation links.

The phrase “accessible through MCP” is deliberately conditional. No universal list of ADT MCP functions applies to every installation. Administrators should publish an environment-specific catalog that identifies each enabled tool, its accepted parameters, its expected output, the target system, and the required authorization.

How IBM Bob uses this information

IBM Bob can use retrieved technical information as grounded context for explanation and analysis. Its role is to help users understand the evidence returned by the configured MCP tools, not to bypass normal SAP development processes or replace accountable technical review.

For code comprehension, IBM Bob can describe the purpose and flow of an authorized ABAP object, identify important methods or routines, and explain data types referenced in the source. For example, after retrieving a class and a related structure, it can explain how a method maps input fields to an internal representation.

For technical analysis and impact assessment, IBM Bob can organize a starting inventory of related artifacts. If a team is considering a change to a table field, it can help enumerate visible references, locate relevant source objects, and suggest questions for a human reviewer: Are there interfaces that consume this field? Are there conversion routines? Are dynamic references or configuration dependencies outside the retrieved set?

For incident troubleshooting, IBM Bob can help translate an incident symptom into a targeted inspection sequence. A report that an outbound message is missing a value could lead to retrieval of the interface structure, the mapping class, and the related table definition. The resulting explanation can guide a support analyst toward the next diagnostic steps, such as reproducing the issue, checking application logs, or reviewing runtime data in approved tools.

For guided development, legacy-code understanding, onboarding, and managed services, IBM Bob can summarize authorized technical artifacts in accessible language and highlight terms that new team members should investigate. This can reduce the time required to form an initial mental model. It does not remove the need for coding standards, peer review, test execution, change control, or knowledge from functional owners.

Key benefits

A carefully governed ADT MCP integration can provide several practical benefits:

- **Faster initial orientation:** Engineers and support personnel can begin with focused repository evidence rather than broad manual searches.
- **More consistent technical conversations:** Structured retrieval gives teams a shared basis for discussing object definitions, packages, and visible relationships.
- **Improved accessibility of legacy knowledge:** Clear explanations of existing objects can help personnel become productive in systems with long histories and limited documentation.
- **Support for evidence-led analysis:** Retrieved source and metadata can inform impact assessments and incident investigation, while keeping verification steps explicit.
- **A reusable controlled interface:** Defined MCP tools can establish repeatable access patterns instead of relying on ad hoc data copying or ungoverned connections.

These benefits depend on accurate configuration, appropriate permissions, a maintained tool catalog, and user discipline. An integration that returns incomplete information, uses over-privileged credentials, or lacks auditability can introduce risk rather than value.

Security considerations

Security should be designed into the integration rather than added after connectivity is established. The central principle is least privilege: the MCP server and its SAP identity should have only the access required for the intended inspection use cases.

Where the objective is repository understanding, begin with read-oriented ADT services and avoid enabling change-capable operations unless there is a separately approved business requirement. Access should be scoped by system, client, repository area, and object authorization where the SAP authorization design supports those boundaries. Production use requires particular scrutiny because source and metadata can reveal sensitive implementation details even when no business data is retrieved.

Credentials, tokens, certificates, and service URLs must be protected as secrets. Store them in approved secret-management or platform configuration facilities; do not place them in prompts, source control, screenshots, or user-facing examples. Use encrypted transport and validate certificates according to organizational policy. Limit network paths so that only approved components can reach the MCP server and SAP endpoints.

Logging and monitoring should make it possible to understand which tool was called, by whom or on whose behalf, against which target, and whether the call succeeded. Logs should be designed to avoid unnecessarily storing source code or sensitive values. Retention, review, and incident-response procedures should align with the organization's security and compliance requirements.

Finally, validate the boundary between retrieval and action. IBM Bob can summarize retrieved information or recommend a next step, but it should not be treated as an authority to deploy code, alter configuration, approve transports, or make production changes. Those actions remain subject to established SAP controls and accountable human review.

Business impact

The primary business impact is improved responsiveness in technical work that depends on accurate repository context. Support teams can more quickly establish what an object is, where it belongs, and what related definitions warrant review. Development teams can reduce time spent assembling basic context before design discussions. Managed-service providers can use the integration to create clearer handoffs and more repeatable investigative workflows.

The impact should be measured with operational indicators rather than assumed. Useful measures can include time to triage a technical incident, time for a new team member to locate and explain a repository artifact, number of

manual tool-navigation steps in a standard investigation, and the quality of evidence attached to an impact assessment. Baselines should be collected before broad rollout where practical.

The integration does not guarantee faster resolution of every issue. Root cause can depend on runtime state, configuration, authorization, external systems, data quality, and business process context that source inspection alone cannot establish. Its value is strongest when it helps teams ask better questions and reach the right verification steps sooner.

Example scenario

Consider a support team investigating an issue in which an outbound purchasing interface occasionally omits a supplier reference. The incident report identifies the message type but not the ABAP implementation responsible for mapping the value.

An authorized engineer asks IBM Bob to help identify the relevant repository artifacts. Using only the MCP tools that are configured and authorized, IBM Bob retrieves the visible interface structure definition and searches the applicable package for related mapping objects. The returned information shows a structure field for the supplier reference and identifies a class whose method populates the outbound payload.

IBM Bob explains the apparent mapping path in plain language: the method reads a value from an internal structure, applies a conditional check, and assigns it to the outbound field. It also notes an important limitation: the retrieved code alone does not show whether the input value was present at runtime, whether a configuration rule selected a different path, or whether a downstream system discarded the value.

The support engineer uses this result as a guided investigation plan. They review the relevant source in the approved development environment, reproduce the case with permitted test data, inspect application and interface logs, and confirm whether the condition is reached. If a code change is needed, the team follows its normal design review, test, transport, and release process. In this scenario, MCP-based inspection accelerates understanding without granting uncontrolled change authority.

Summary

An SAP ADT MCP server can connect IBM Bob to selected SAP development-time information through configured ADT services. The integration can support structured retrieval of authorized ABAP source, package and object information, dictionary definitions, metadata, technical relationships, and other repository details made available by the environment.

The correct operating model is controlled remote inspection. IBM Bob uses MCP tools to gather and explain evidence; the MCP server constrains the available interface; and SAP services and authorizations determine what can actually be retrieved. Service availability, tool behavior, object visibility, and permissions vary by implementation and must be verified in each landscape.

With least-privilege access, secure configuration, monitoring, and continued human oversight, this approach can help teams comprehend code, assess technical change, troubleshoot incidents, support legacy applications, onboard personnel, and deliver managed services more consistently. It should complement—not replace—established SAP engineering, security, and change-management practices.

Integrating abapGit with IBM Bob for governed AI-assisted SAP ABAP development (SAP ADT MCP server)

Enterprises adopting AI-assisted software development for SAP environments face a fundamental governance question: when code can now be generated or modified by an AI-assisted tool, by a developer working in Eclipse or ABAP Development Tools (ADT), and by developers collaborating across geographies, how does the organization maintain a single source of truth, a complete audit trail, and a controlled path to production?

This section presents a reference architecture and step-by-step implementation guide for integrating IBM Bob, an AI-assisted development environment for SAP, with abapGit and a Git-based source-control platform such as IBM GitHub Enterprise or an equivalent client Git repository. The resulting pipeline extends established DevOps practices

— branching, pull requests, code review, and merge-conflict resolution — into SAP ABAP development, regardless of whether the code originates from a human developer or an AI-assisted tool.

For senior management and program leadership, the business case is straightforward: uncoordinated, AI-accelerated code changes introduce risk to transport consistency, deployment stability, and delivery timelines. A governed Git-based pipeline mitigates this risk while preserving the productivity gains that AI-assisted development delivers. For architects and technical leads, this section defines the target architecture, the required SAP and GitHub configuration, and the controls that should be embedded in the pipeline. For developers, it provides a detailed, reproducible procedure — from initial one-time setup through day-to-day use — for creating and modifying ABAP objects with IBM Bob and deploying them safely into an SAP system.

Business challenge

Consider a common scenario in a multi-developer SAP landscape: one developer uses an AI-assisted development tool, such as IBM Bob, to generate ABAP code, while a second developer simultaneously modifies the same object directly in Eclipse or ADT. Several questions immediately arise:

- Which version of the object is authoritative?
- How are the two sets of changes tracked?
- How are conflicting changes identified and resolved?

In most SAP landscapes today, these questions cannot be answered with certainty. In a multi-developer environment, unmanaged concurrent changes introduce material operational risk. The consequences extend well beyond isolated coding defects; they can include transport inconsistencies, failed deployments, missed go-live dates, and extended troubleshooting effort to isolate the root cause of a defect.

The introduction of AI-assisted development compounds this challenge rather than resolving it. Code generated within IBM Bob initially exists only in the developer's local workspace. No SAP object lock is applied automatically, and other developers are not notified that work is already under way on the same object. Native SAP tools, in turn, provides no built-in mechanism to compare, merge, or reconcile AI-generated changes against changes made independently in ADT or Eclipse.

Without a centralized, version-controlled process, development activity lacks transparency and governance: traceability is limited, audit history is incomplete, and there is no reliable mechanism for coordinating concurrent modifications. Transport coordination is left to manual communication and spreadsheet-based tracking, which increases the likelihood of error and reduces overall delivery efficiency. This model does not scale to large SAP environments, and it becomes progressively less sustainable as AI-assisted development is adopted more broadly across the organization.

Solution overview

Modern SAP developers work across multiple interfaces — Eclipse, ADT, traditional SAP back-end transactions, and, increasingly, AI-assisted platforms such as IBM Bob. IBM Bob improves developer productivity by generating code, refactoring legacy ABAP, and implementing enhancements in line with the project's defined standards, skills, and governance policies. To capture this productivity benefit without sacrificing control, the AI-assisted workflow must be brought under the same version-control discipline that already governs conventional development.

abapGit provides the governance layer that closes this gap. abapGit is a free, installable open-source tool for the SAP system that enables a structured, version-controlled development workflow. It connects IBM Bob, a Git-based repository — such as IBM GitHub Enterprise or an equivalent client Git repository — and the standard SAP object-activation process into a single, managed pipeline.

Because the entire workflow is Git-based, the organization gains the full set of standard source-control capabilities for ABAP development: branching, merge-conflict resolution, and pull-request-based review. Multiple developers can work concurrently on the same objects — in IBM Bob, in Eclipse, or in ADT — and any conflicts are surfaced and resolved in Git, before changes ever reach the SAP system, rather than being discovered afterward.

End-to-end workflow

The integrated pipeline consists of six steps summarized in this section and illustrated in the following figures.

Step 1: Download

Existing SAP objects are brought into the IBM Bob workspace so that Bob has full context of the current code base. This is done by pushing from abapGit to IBM GitHub (or an equivalent client Git repository) and then pulling that Git repository into the IBM Bob workspace. As a result, IBM Bob always works against the organization's real, current code — not an assumed or outdated version.

Step 2: Develop

The developer works within IBM Bob, an AI-assisted environment governed by the project's defined skills, rules, and custom modes. Code is generated, modified, or refactored with full project context and the appropriate guardrails in place.

Step 3: Stage and commit

After changes are complete, the developer stages them using standard Git operations (`git add`, `git commit`) directly from within IBM Bob — a familiar, traceable step for any developer accustomed to modern source control.

Step 4: Push to Git

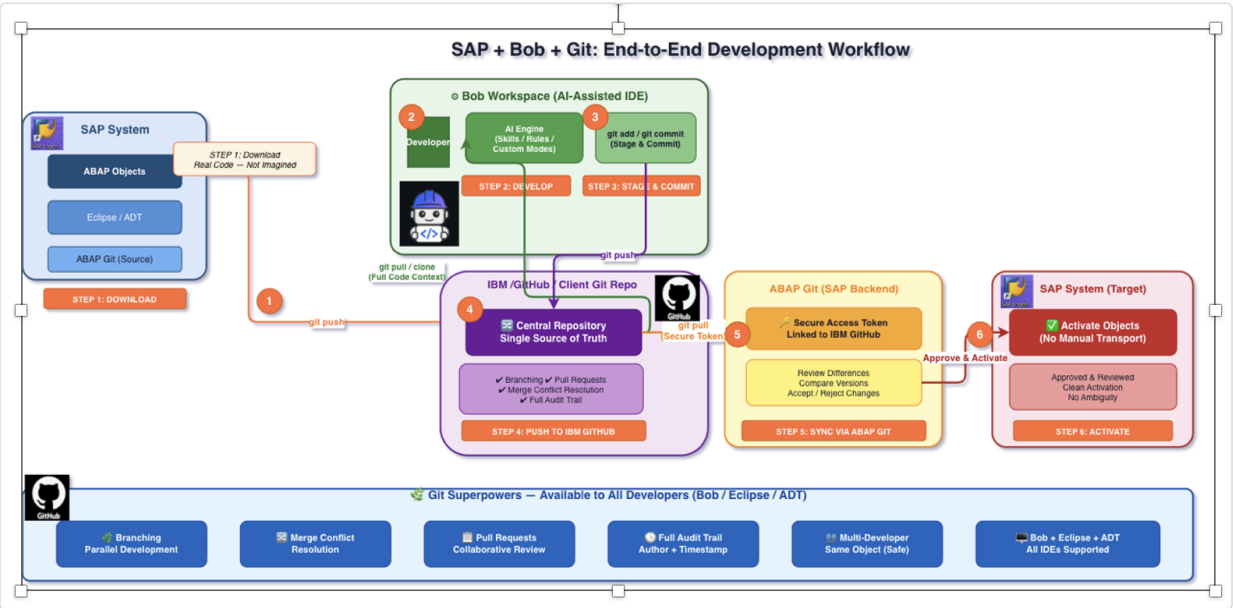
The committed changes are pushed to a central IBM GitHub repository or client Git repository, which becomes the single source of truth for the change. Every change, author, and timestamp is recorded permanently.

Step 5: Synchronize through abapGit

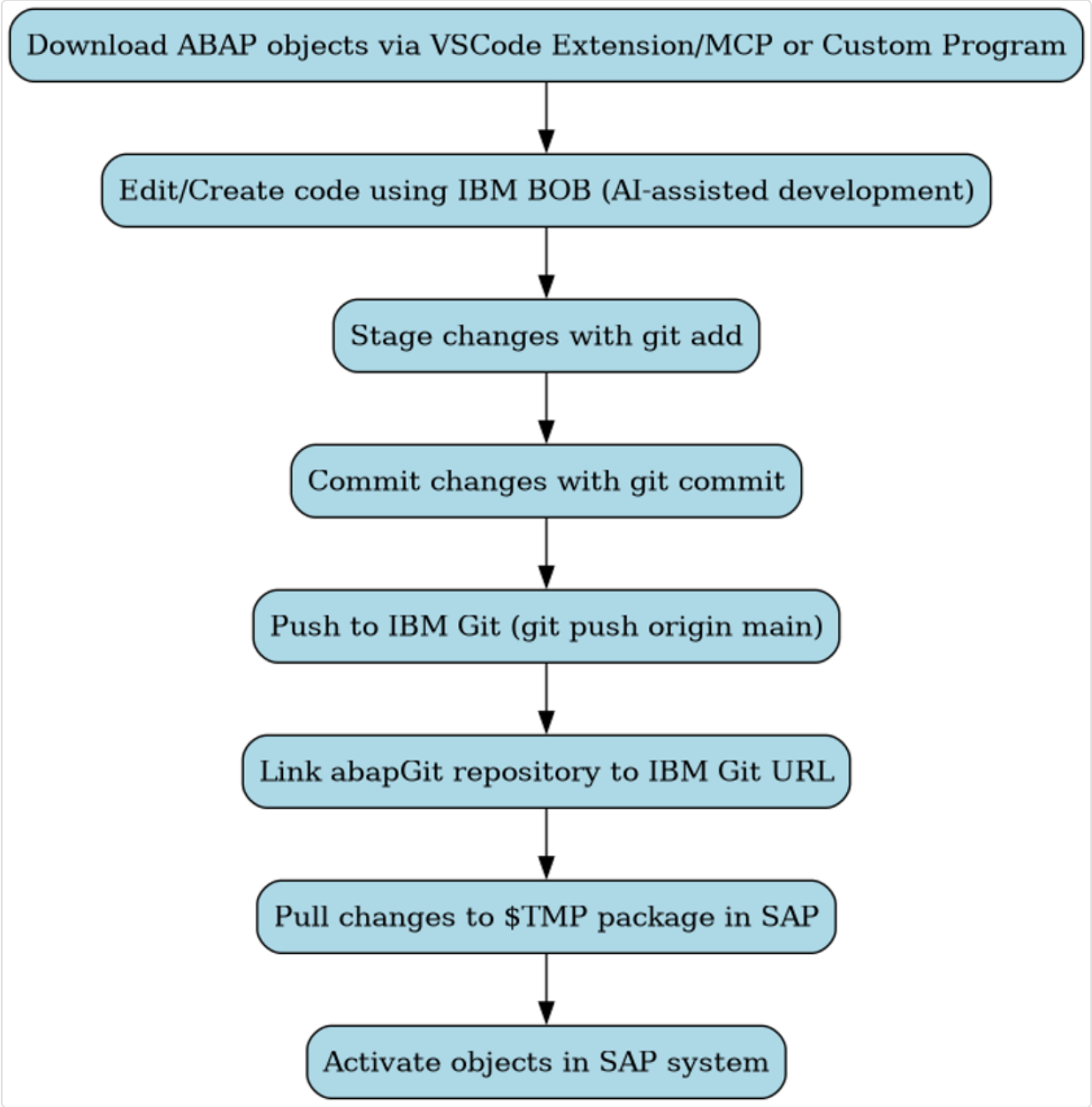
The abapGit installation in the SAP back end is linked to the same Git repository through a secure access token. When new changes arrive, abapGit pulls them into the SAP system, where the team can review differences, compare versions, and accept or reject the changes — entirely within the SAP environment.

Step 6: Activate

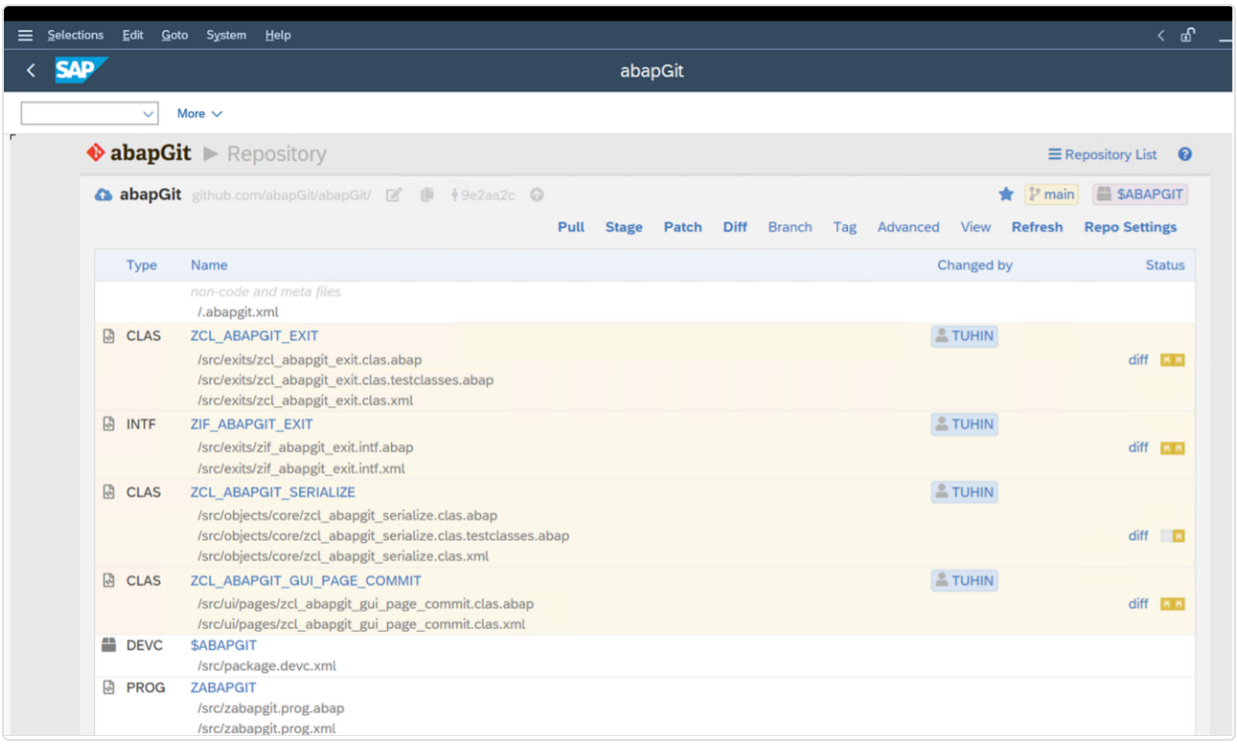
After the changes have been reviewed and approved, the corresponding objects are activated in the SAP system through the standard activation and transport process, without manual transport management or ambiguity about which version is authoritative.



Six-step workflow integrating IBM Bob, Git, and the SAP system



Eight-step abapGit and IBM Bob pipeline



abapGit repository view showing ABAP objects in package ZABAP

Implementation and configuration walkthrough

This section provides the detailed, step-by-step configuration and operating procedure for the integration. It is organized into five phases, summarized in the following table and described in detail in the subsections that follow.

Table: The phases of the abapGit / IBM Bob integration

Phase	Description
1	Configure IBM Bob and Git connectivity (one-time setup)
2	Download ABAP objects from SAP into the IBM Bob workspace
3	Develop locally with IBM Bob
4	Stage, commit, and push changes to Git
5	Deploy changes back to SAP through abapGit

Phase 1 — Configure IBM Bob and Git connectivity (one-time setup)

Phase 1 is performed once per SAP system and Git repository pair. It establishes the foundational connectivity between the SAP back end, the Git repository, and IBM Bob.

Install abapGit in the SAP system

abapGit must be installed in the SAP back end before any objects can be exchanged with a Git repository. Complete the following steps:

1. Navigate to the *abapGit User Guide* for the current installation instructions.

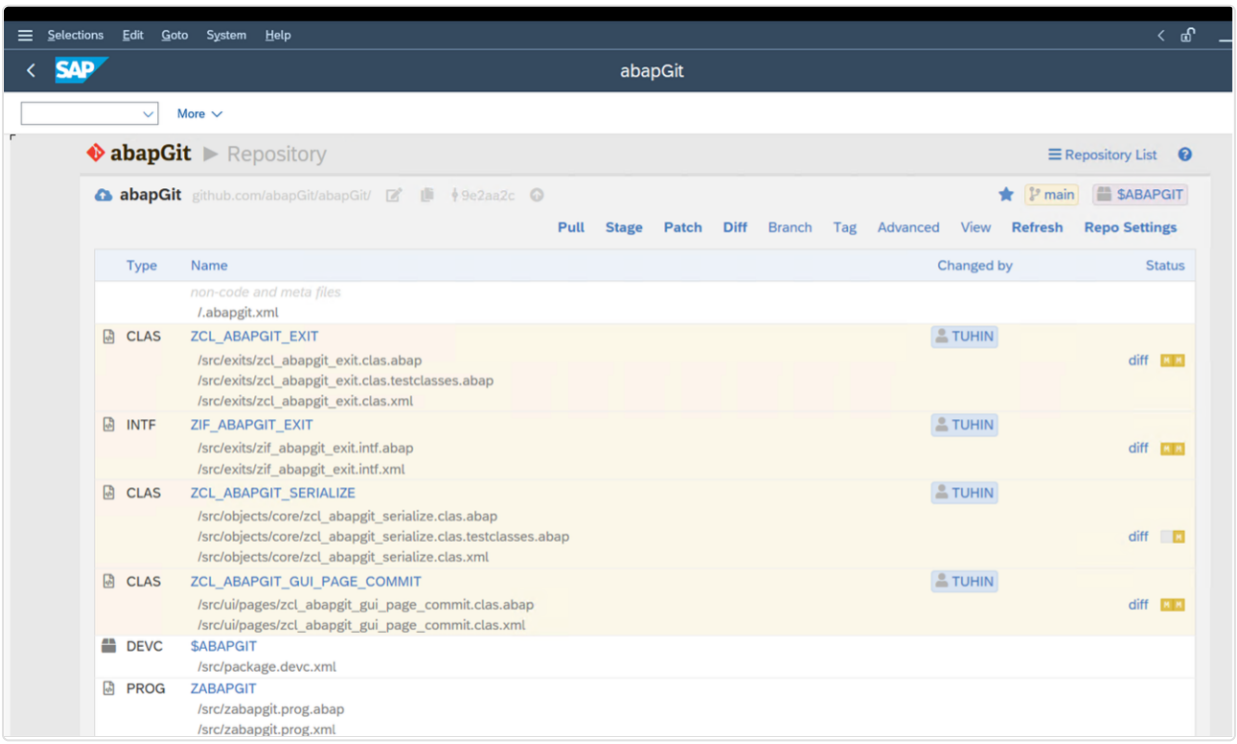
2. Install the stand-alone version (`ZABAPGIT_STANDALONE`) into the SAP system.
3. Verify the installation by running the program and confirming that the abapGit user interface loads correctly.

> **Note:** Confirm that the appropriate authorizations (for example, `S_DEVELOP` and `S_TRANSPRT`) are assigned before proceeding.

After the interface is loaded, complete the following steps:

1. Run the stand-alone version of abapGit.
2. On the repository list screen, select **New Online**.
3. Enter `https://github.com/abapGit/abapGit/` as the repository URL.
4. Enter the target package name (`$ABAPGIT`), or select an existing package, or create a new one.
5. Select **Create Online Repo**.
6. Select **Pull**.

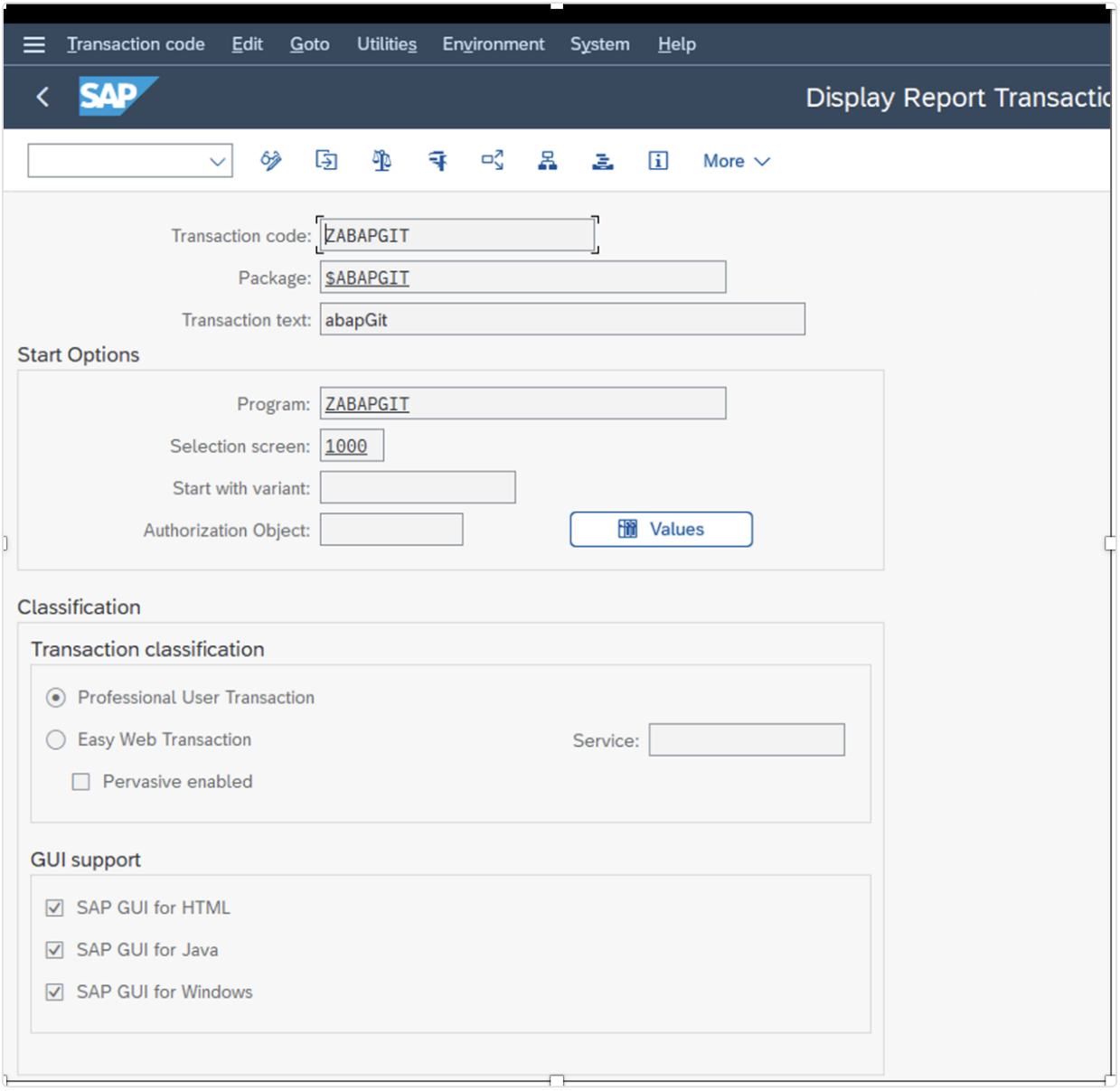
This step installs the complete abapGit repository into the SAP back end and can take several minutes to complete. A successful installation is shown in the following figure.



abapGit repository installed in package $ABAPGIT

> **Note:** Every ABAP object exchanged through abapGit is accompanied by a metadata `.xml` file that defines the object type and related attributes. This file is required for abapGit to correctly classify and process the object; its presence should always be verified alongside the corresponding source file.

After installation is complete, transaction `ZABAPGIT` is available for day-to-day use, as shown in the preceding figure.



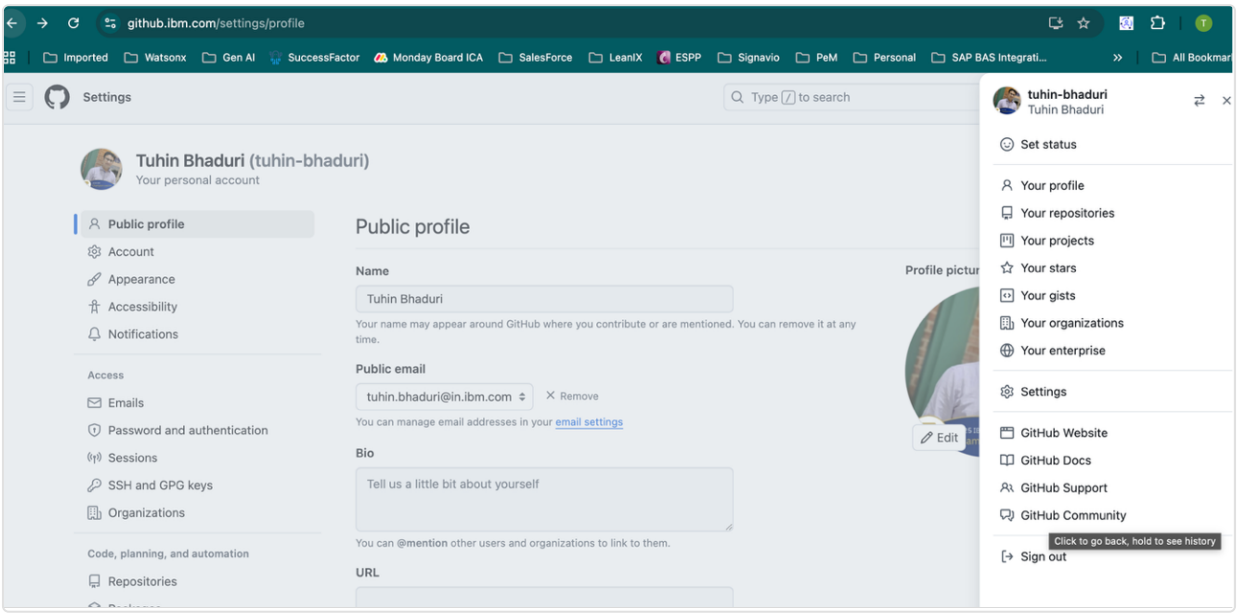
ZABAPGIT transaction properties

Establish secure connectivity to the Git repository

The online abapGit repository must be connected to IBM GitHub Enterprise, or an equivalent client Git repository, using a secure token over SSL. The following steps configure a personal access token for this purpose.

Step 1: Open GitHub settings

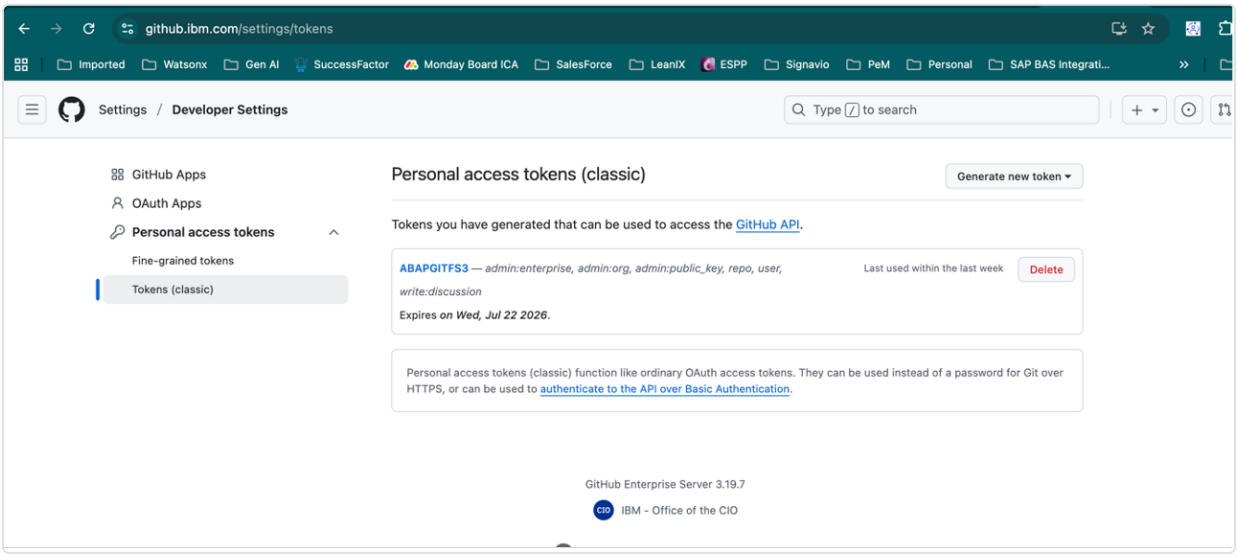
Open the IBM GitHub Enterprise page (or the equivalent client GitHub instance), select the user profile, and choose **Settings**.



IBM GitHub Enterprise profile settings with the user menu open

Step 2: Open developer settings

Select **Developer settings** from the left-hand navigation panel.

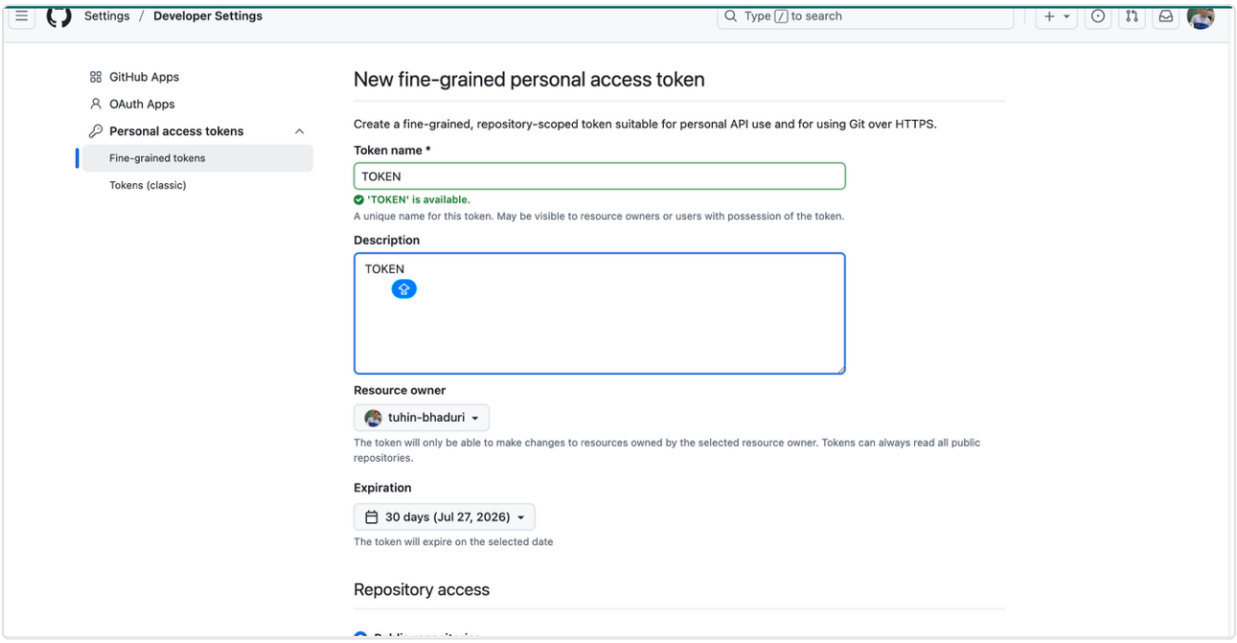


IBM GitHub Developer Settings, personal access tokens (classic)

Step 3: Open the personal access tokens screen

Select **Tokens (classic)** from the left-hand navigation panel to open the Personal Access Tokens screen.

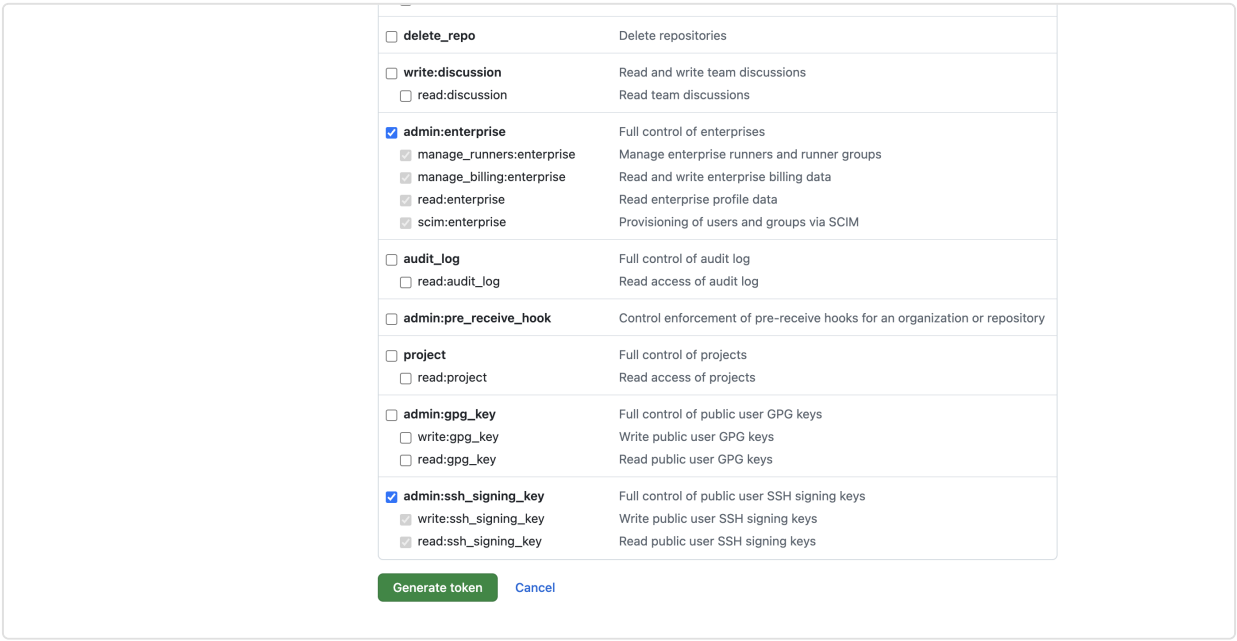
Click **Generate new token**.



GitHub new fine-grained personal access token form

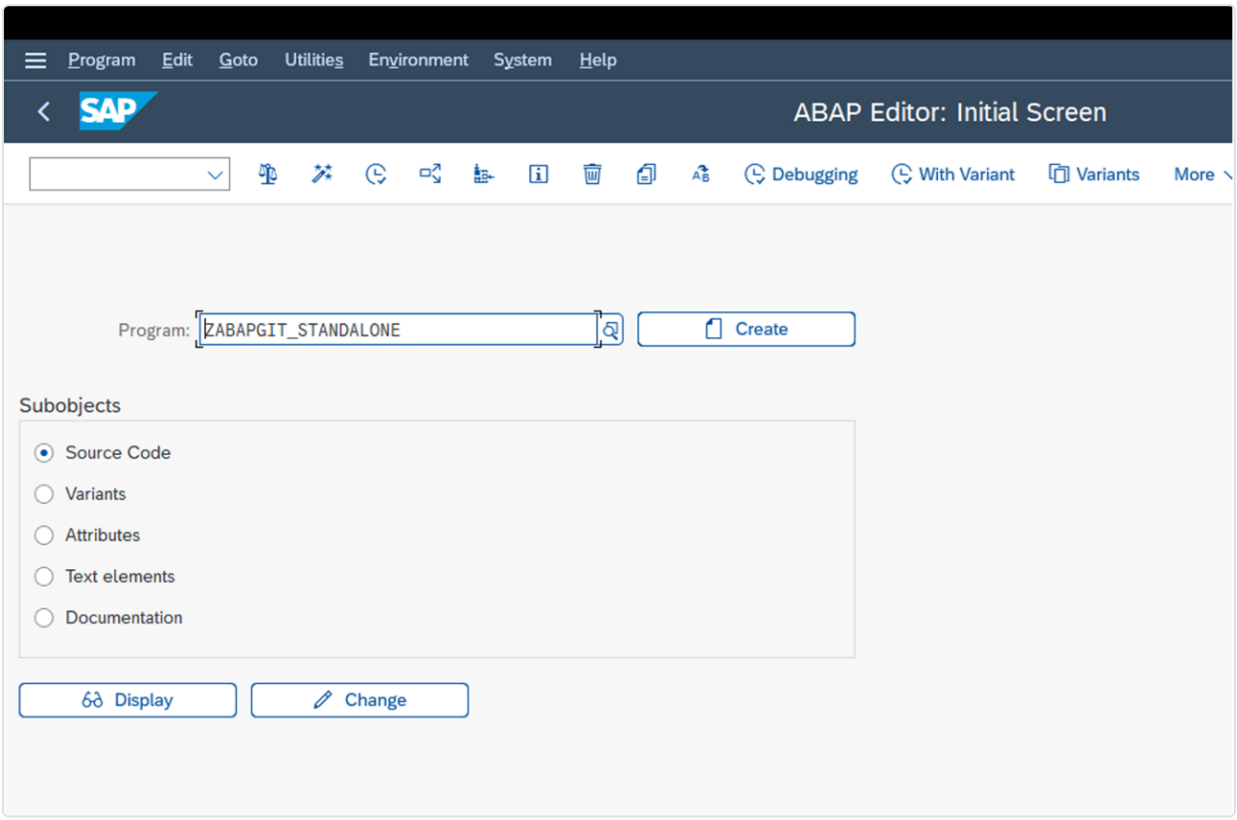
Step 4: Generate and record the token

Select **Generate new token**, provide a token name and description, and generate the token. Record the resulting alphanumeric token; it is required when configuring the abapGit online repository.



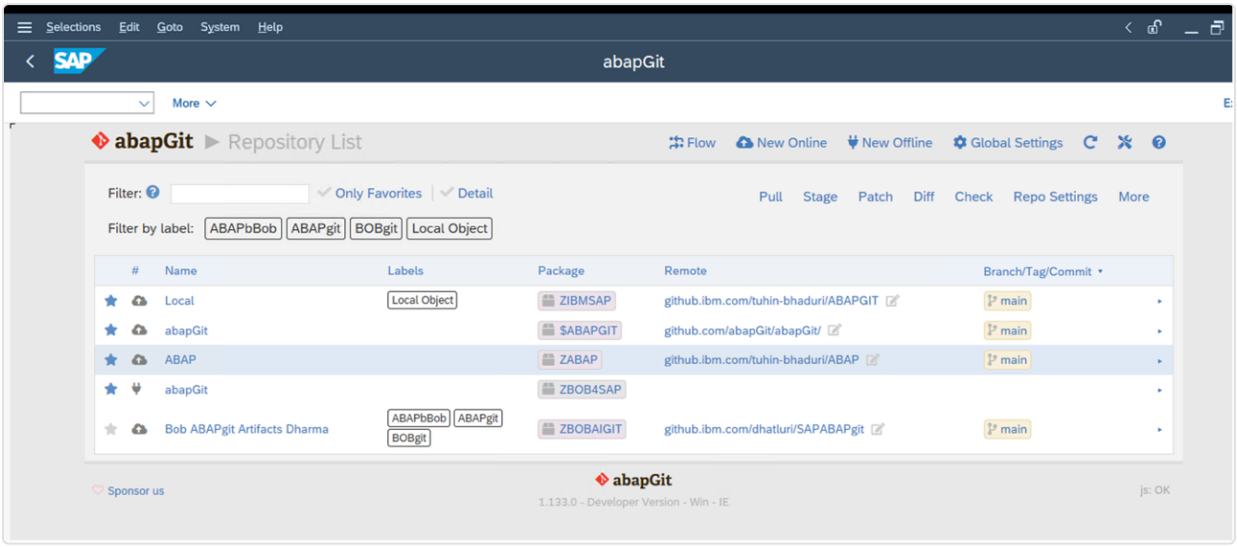
GitHub token scope selection page with Generate token button

Return to the SAP system, start transaction `SE38`, and run the abapGit program installed in the previous step.



SE38 with ZABAPGIT_STANDALONE selected for execution

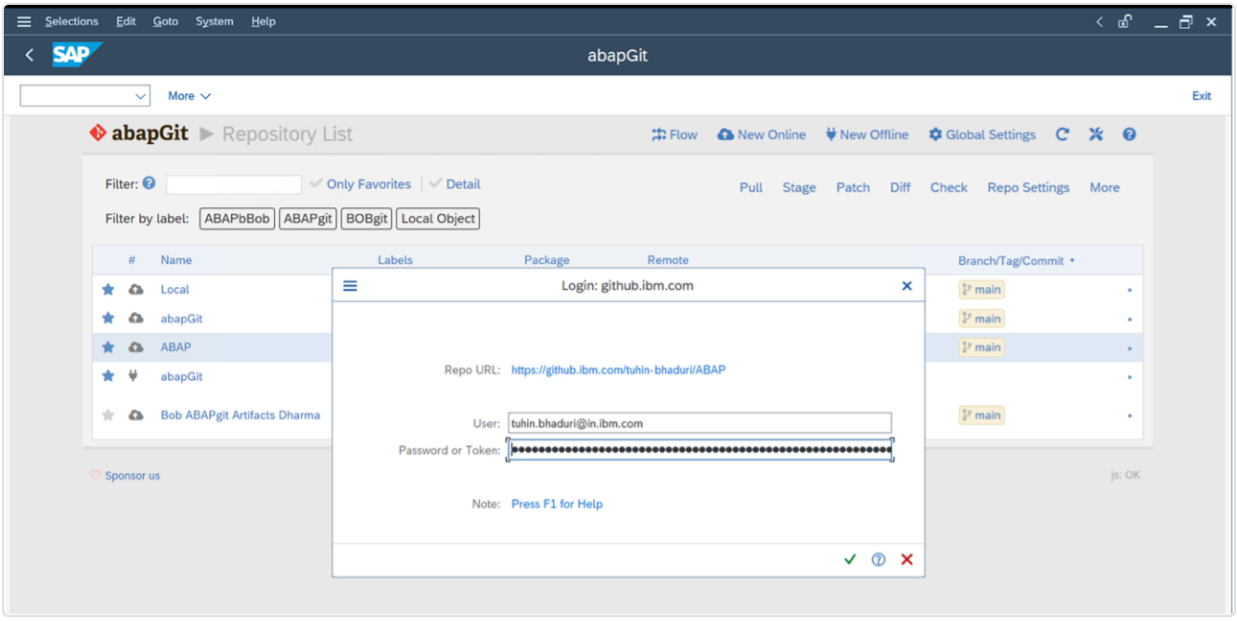
Running the program opens the abapGit user interface, shown in the following figure.



abapGit repository list in SAP

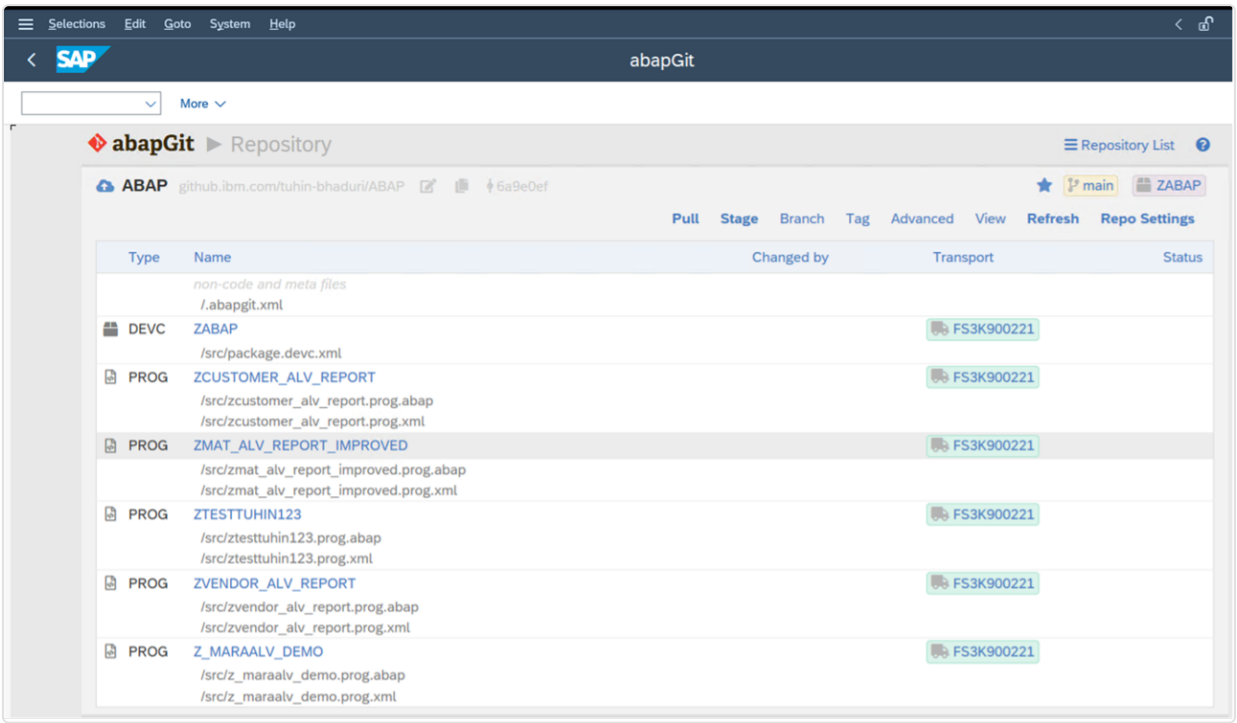
Select **Create Online Repository**, as shown in the preceding figure, and provide the ABAP object details. abapGit prompts for the password or token created in the preceding step.

> **Note:** Depending on the security configuration of the enterprise GitHub instance, the token might be requested more than once during setup. Behavior might differ slightly when using a public GitHub account rather than an enterprise instance.



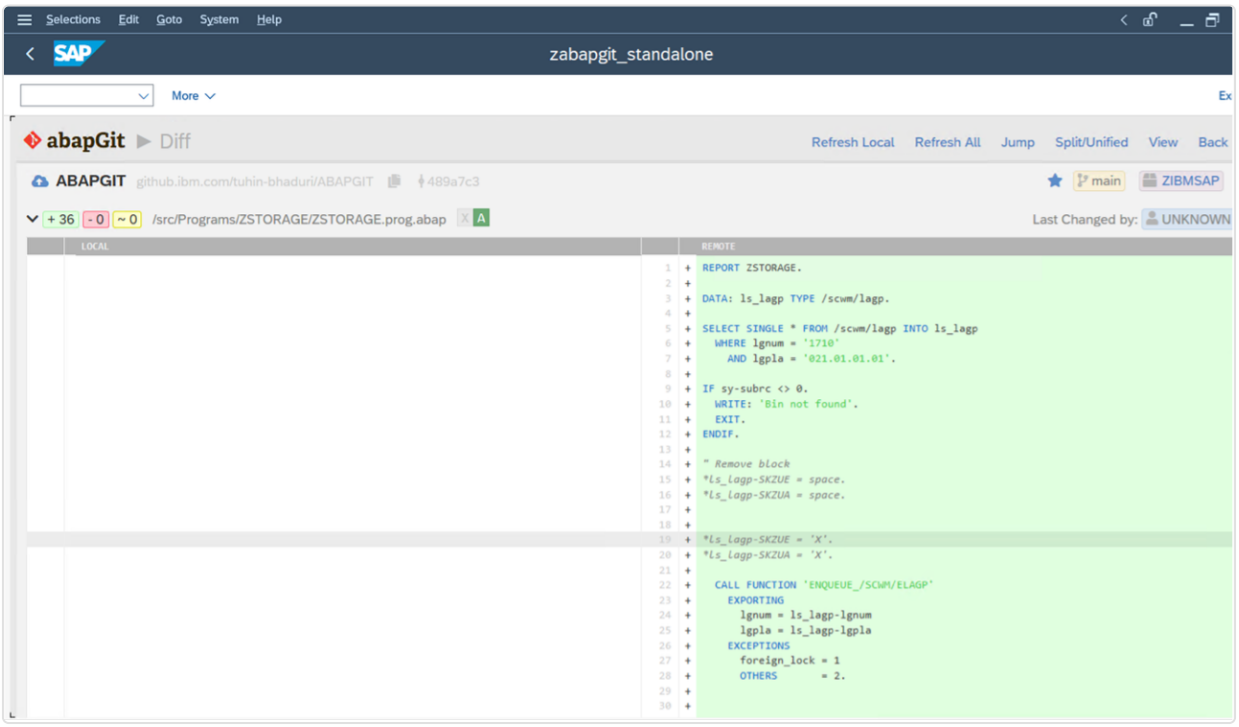
abapGit login dialog for the IBM GitHub username and token

After the correct token is supplied, the online repository opens, and objects can be staged from, or pulled into, the connected Git repository — for example, https://github.ibm.com/\<org>/ABAPGIT.git.



abapGit online repository view for package ZABAP

The team can select any ABAP object to compare differences between the local SAP object and the corresponding version in the Git repository.



abapGit diff view of ZSTORAGE.prog.abap with 36 added lines

Phase 2 — Download ABAP objects from SAP

ABAP source objects can be brought into the local IBM Bob workspace using either of the following two approaches. Teams should select the option that best fits their connectivity and security constraints.

Option A — Custom ABAP Download Program

Where direct connectivity between IBM Bob and the SAP system is not available, you can use a custom ABAP program to export objects to the local file system. Complete the following steps:

1. Create a custom ABAP program in transaction SE38 (for example, ZDOWNLOAD_ABAP_OBJECT).
2. Implement the program to serialize the ABAP object using CL_ABAPGIT_OBJECT or an equivalent standard export utility.
3. Save the output to a local folder accessible to IBM Bob.

> **Note:** A sample implementation of this program is provided in Appendix A of this publication.

Option B — Synchronize through abapGit

To synchronize through abapGit, complete the following steps:

1. Push the required changes from abapGit to the IBM GitHub online repository.
2. Clone the repository from IBM GitHub into the IBM Bob workspace.
3. The corresponding files are now available in the IBM Bob workspace.

Phase 3 — Develop locally with IBM Bob

After the ABAP source files are available in the IBM Bob workspace, development proceeds as follows:

1. Open the SAP ABAP source files within IBM Bob.
2. Use the AI-assisted development capabilities of IBM Bob to refactor, extend, or correct existing ABAP code; to generate new ABAP logic, classes, or function modules; and to validate the resulting logic against ABAP best practices.
3. Save all changes locally before proceeding to the Git operations described in Phase 4 — Stage, commit, and push changes to Git.

Phase 4 — Stage, commit, and push changes to Git

Changes can be pushed from IBM Bob to the IBM GitHub online repository by instructing Bob to do so — for example, with the prompt “Push the changes from Bob to IBM Git.” IBM Bob performs the underlying Git operations on the developer's behalf. Developers who prefer to work directly with Git commands can instead follow the following manual procedure.

Create a remote Git repository

Complete the following steps:

1. Log in to the IBM (or client) GitHub Enterprise instance, for example `https://github.ibm.com`.
2. Create a new repository, for example `https://github.ibm.com/<your-org>/ABAPGIT.git`.
3. Record the repository URL for use in the following step.

Set the remote origin

From the IBM Bob workspace terminal, link the local workspace to the remote repository:

```
git remote add origin https://github.ibm.com/<your-org>/ABAPGIT.git
```

Verify that the remote has been configured correctly:

```
git remote -v
```

Stage changes

Add the modified or newly created ABAP source files to the Git staging area:

```
git add .
```

Alternatively, stage specific files individually:

```
git add <path/to/abap-object.abap>
```

Commit changes

Commit the staged changes with a clear, descriptive message:

```
git commit -m "feat: <brief description of ABAP change>"
```

Push to the remote repository

Push the committed changes to the remote Git repository:

```
git push origin main
```

> **Tip:** If this is the first push from the workspace, use git push -u origin main to establish the upstream tracking branch.

Phase 5 — Deploy changes to SAP through abapGit

Open abapGit in SAP

Complete the following steps:

1. Log in to the target SAP back-end system.
2. Start transaction ZABAPGIT.

Open the online repository

Open the online repository created during Phase 1.

Pull, compare, and activate

Complete the following steps:

1. Select Pull to fetch the latest changes from the remote Git repository.
2. Use the Compare view to review the differences between the Git source and the current SAP system objects.
3. Stage the changes to be activated and confirm the pull.
4. The ABAP objects are updated in the SAP system, following the organization's standard transport process.

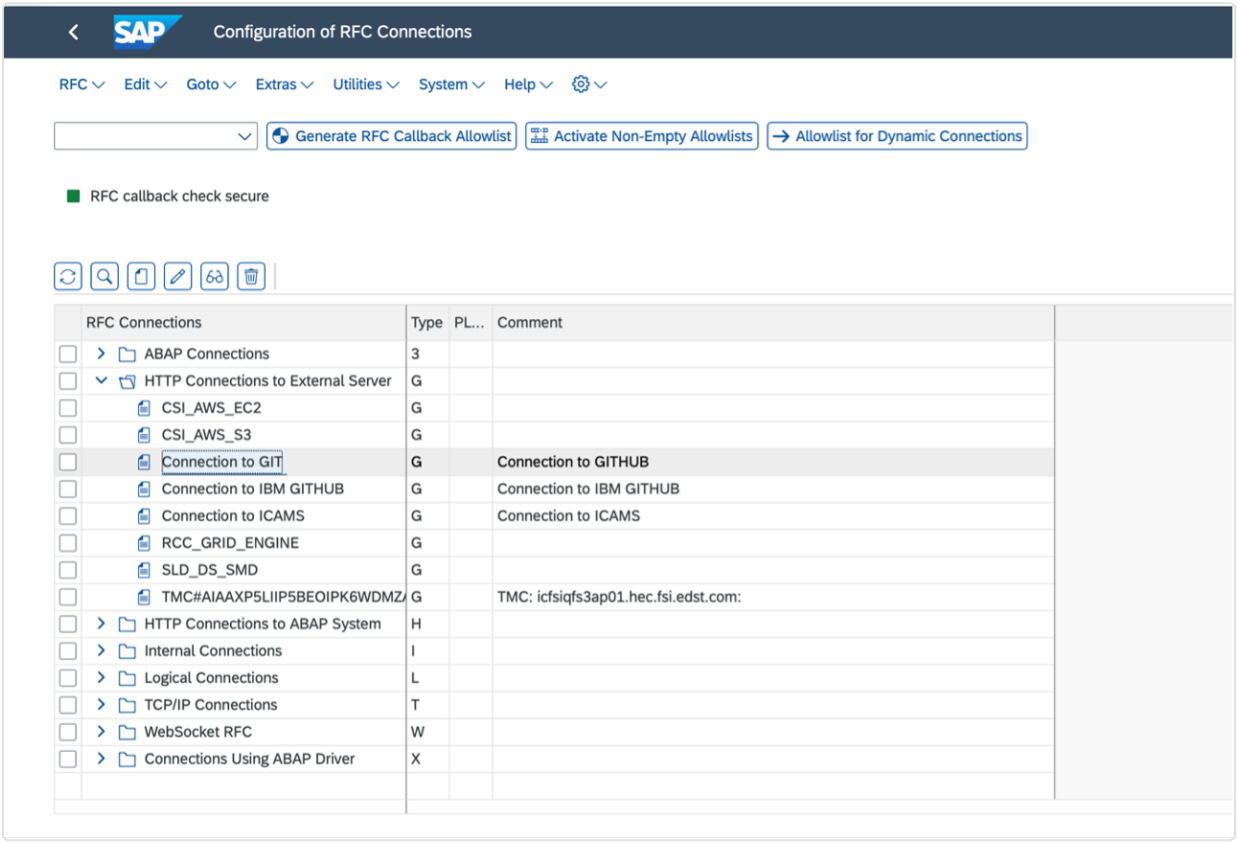
Configuring SAP connectivity to GitHub

This section describes the required RFC destination and proxy configuration to enable connectivity between the SAP system and both public GitHub and the enterprise IBM GitHub instance.

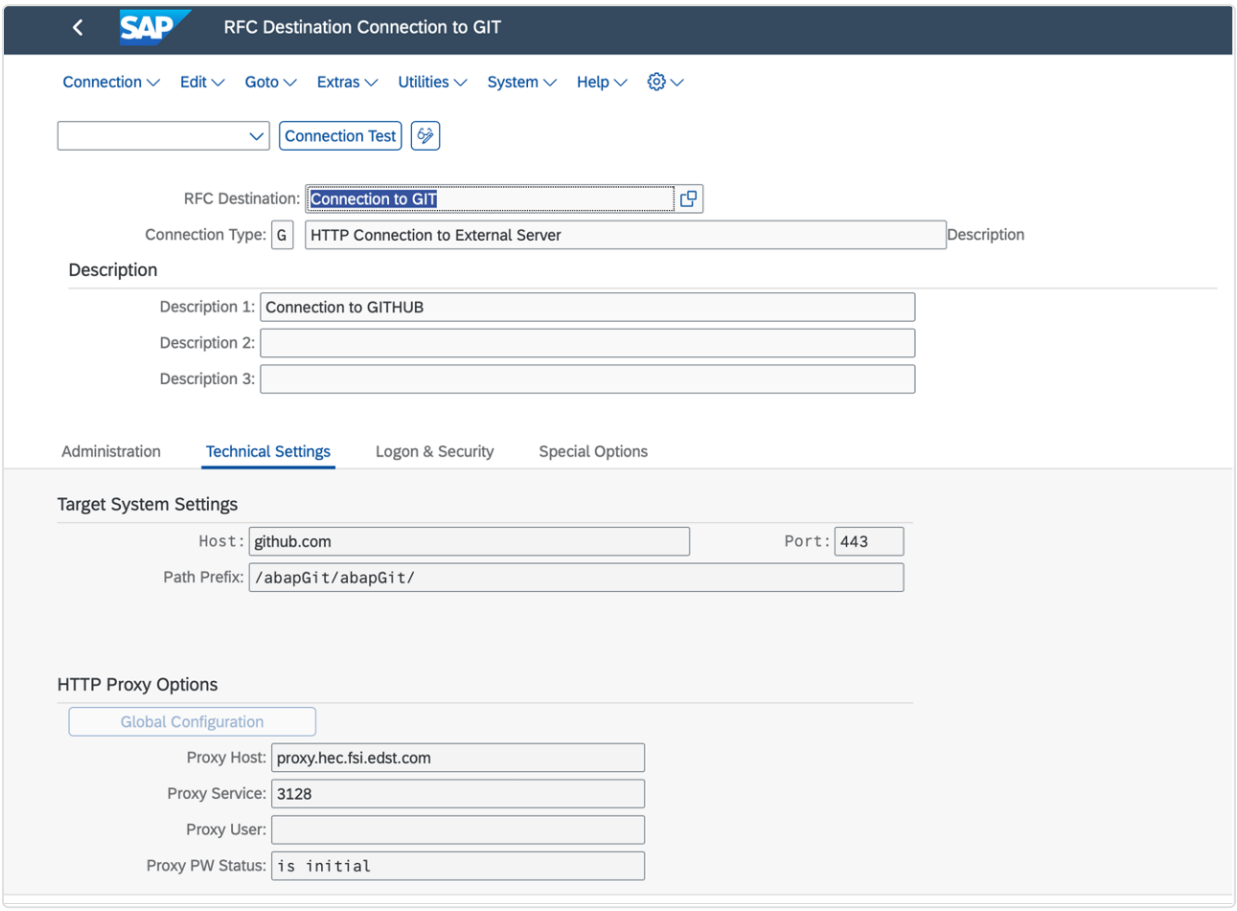
RFC destination for public GitHub

Configure an RFC destination for global connectivity to the public abapGit repository on GitHub:

1. In transaction SM59, create a new RFC destination named, for example, Connection to GIT.
2. Set the target host to github.com and the path prefix to /abapGit/abapGit/.

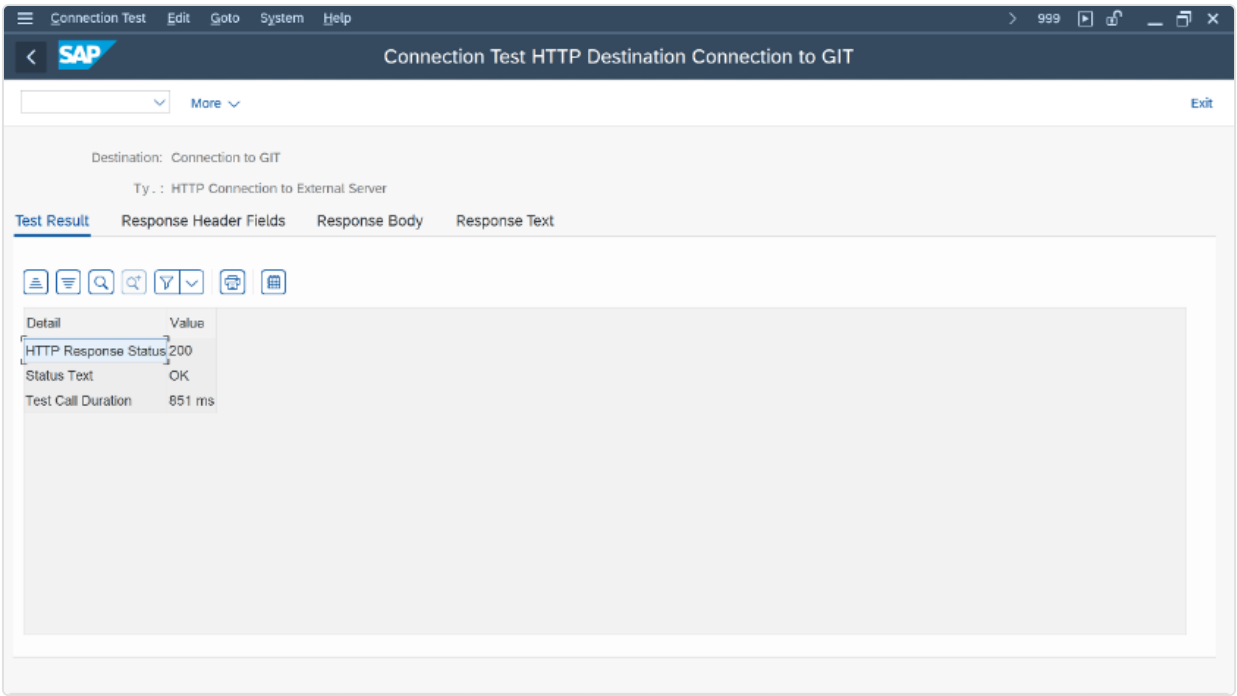


SM59 RFC connections for GitHub and IBM GitHub

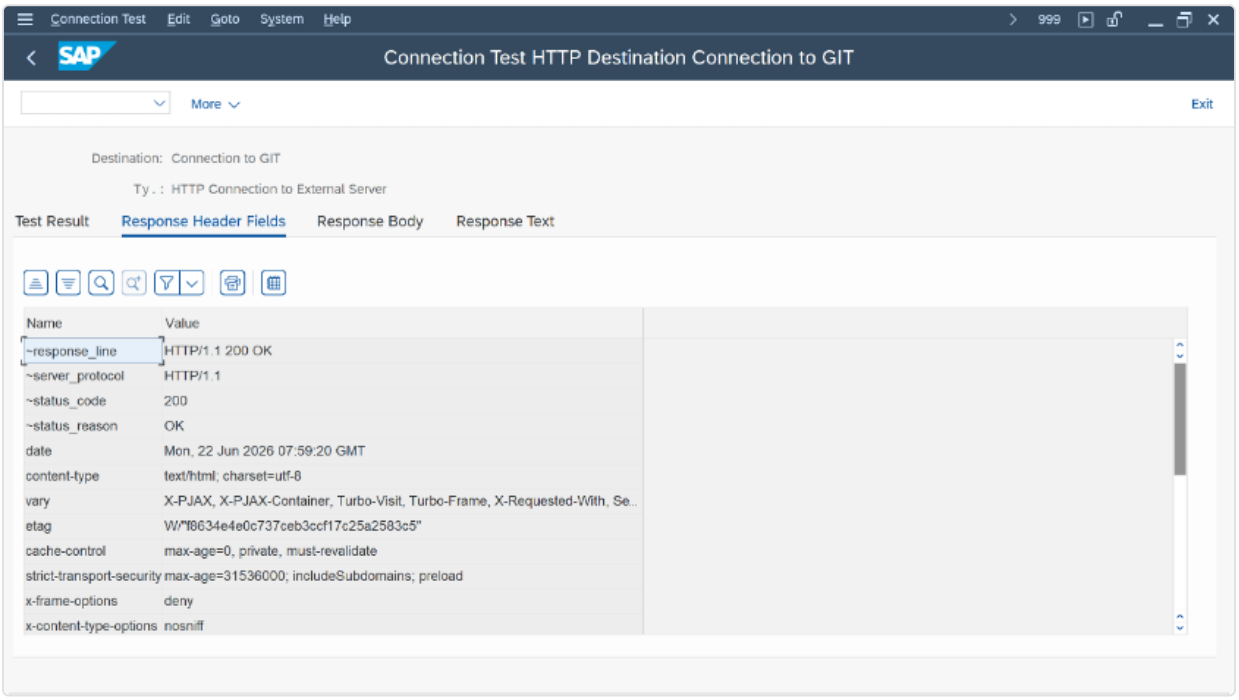


SM59 destination Connection to GIT, technical settings

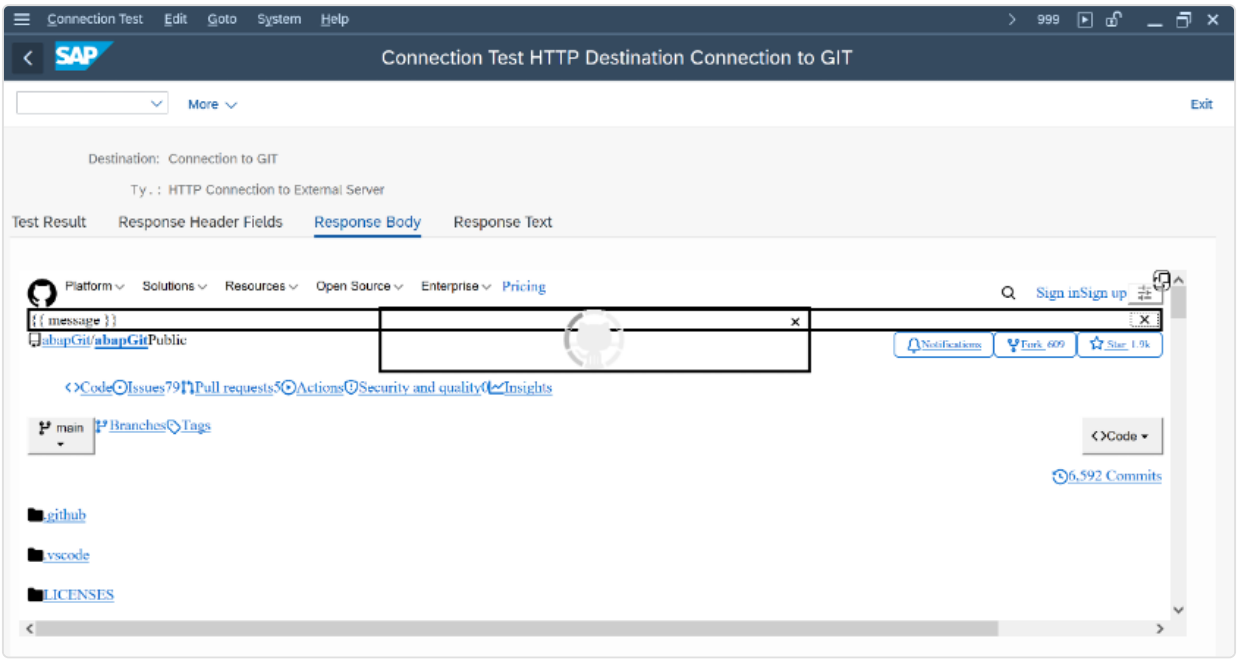
Run the connection test to confirm connectivity to public GitHub, as shown in the following figures.



Connection test result for Connection to GIT



Response header fields for Connection to GIT



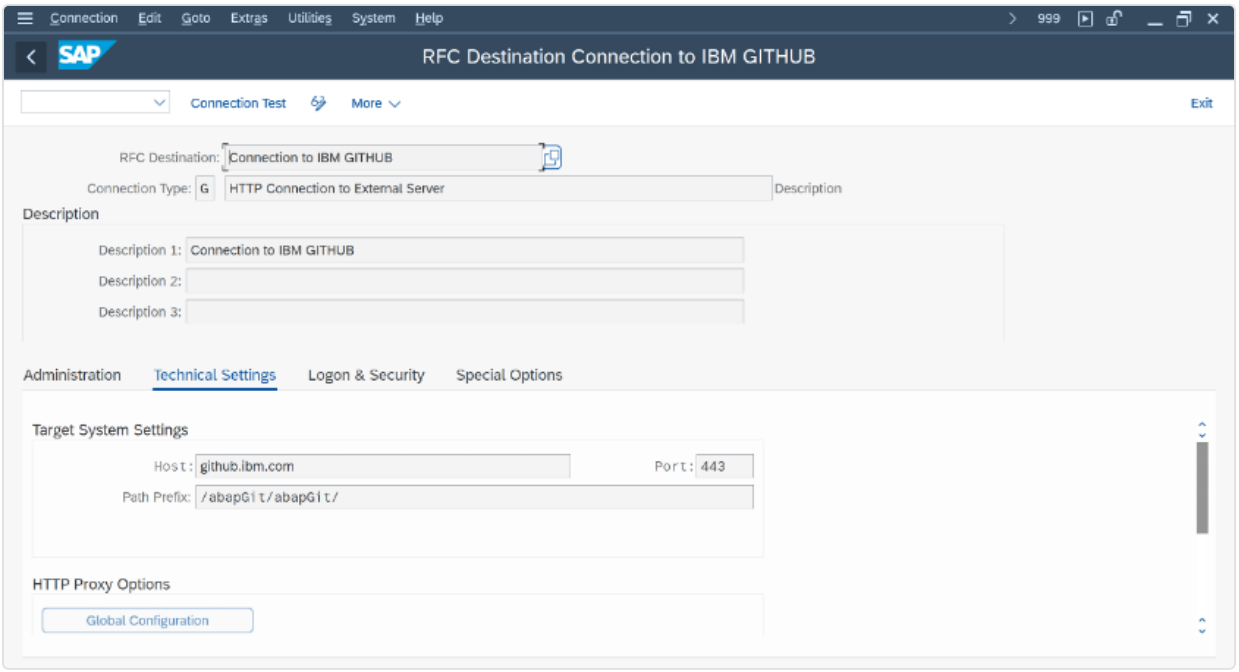
Response body for Connection to GIT



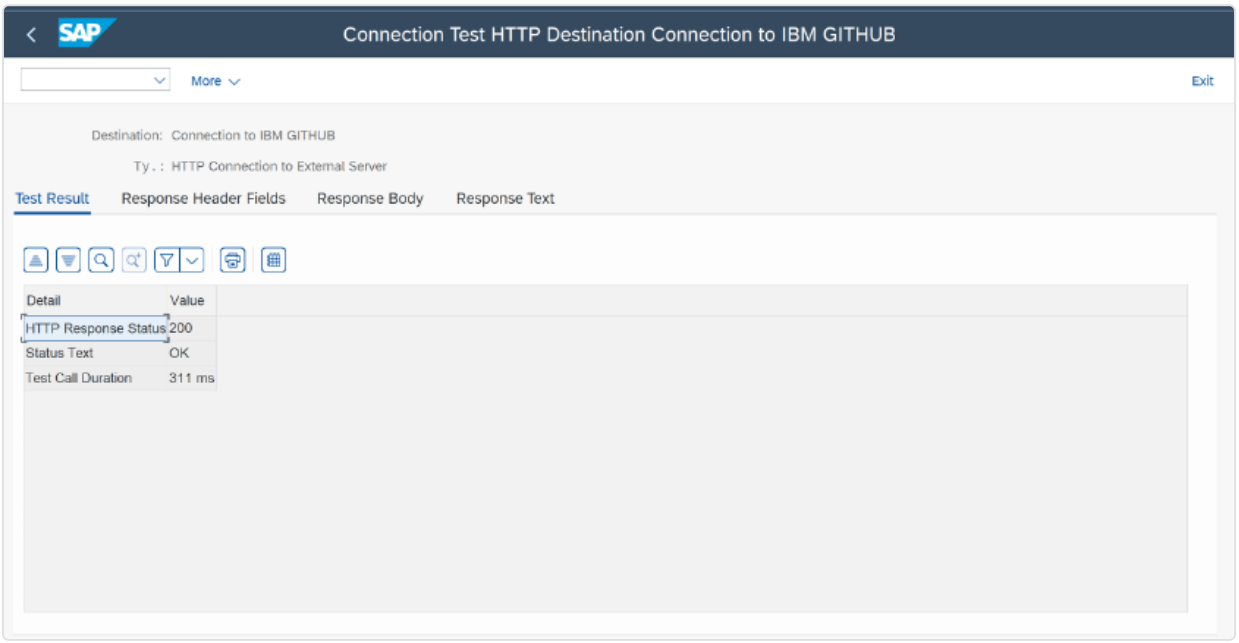
Response text for Connection to GIT

RFC destination for IBM GitHub

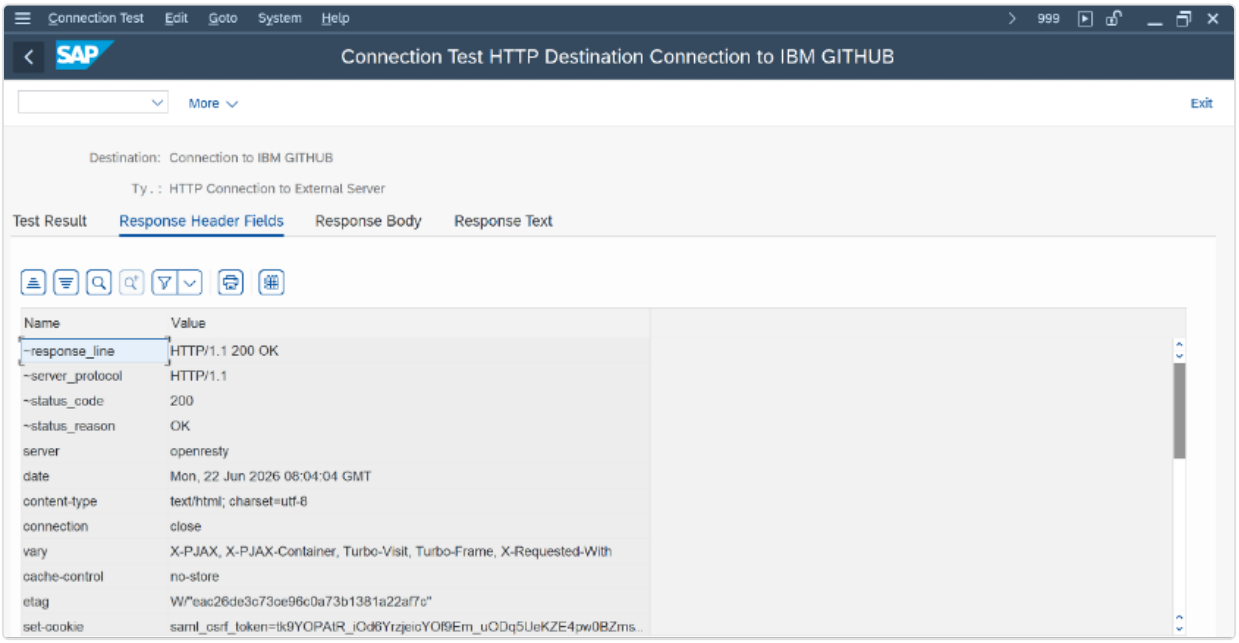
Repeat a similar configuration in SM59 to establish connectivity between IBM Bob and the enterprise IBM GitHub instance, using an RFC destination named, for example, Connection to IBM GITHUB.



SM59 destination Connection to IBM GITHUB, technical settings

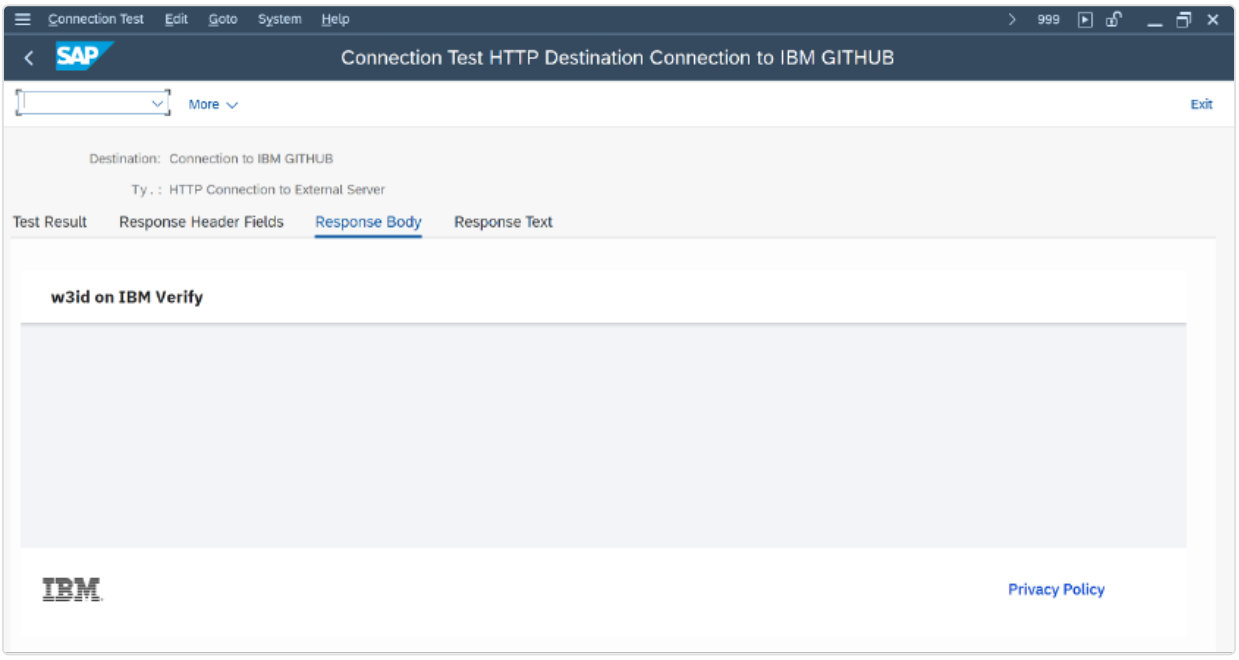


Connection test result for Connection to IBM GITHUB

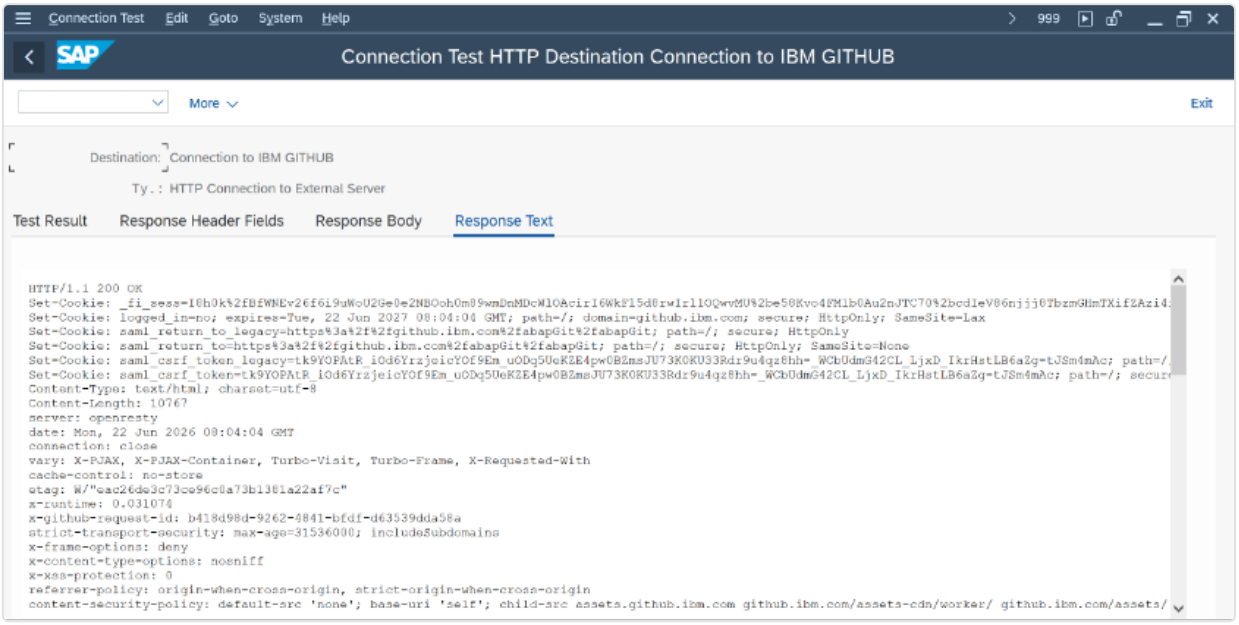


Response header fields for Connection to IBM GITHUB

Test the connectivity to confirm the destination is functioning correctly, as shown in the following figures.



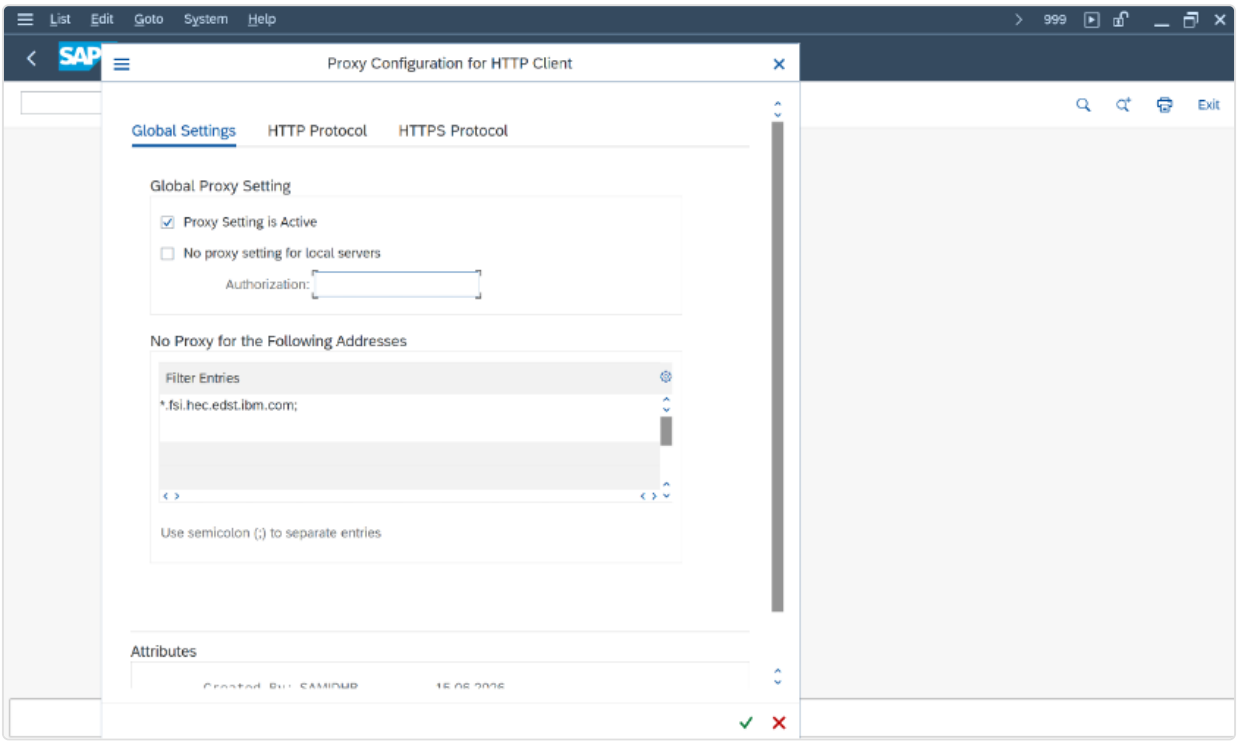
Response body for Connection to IBM GITHUB



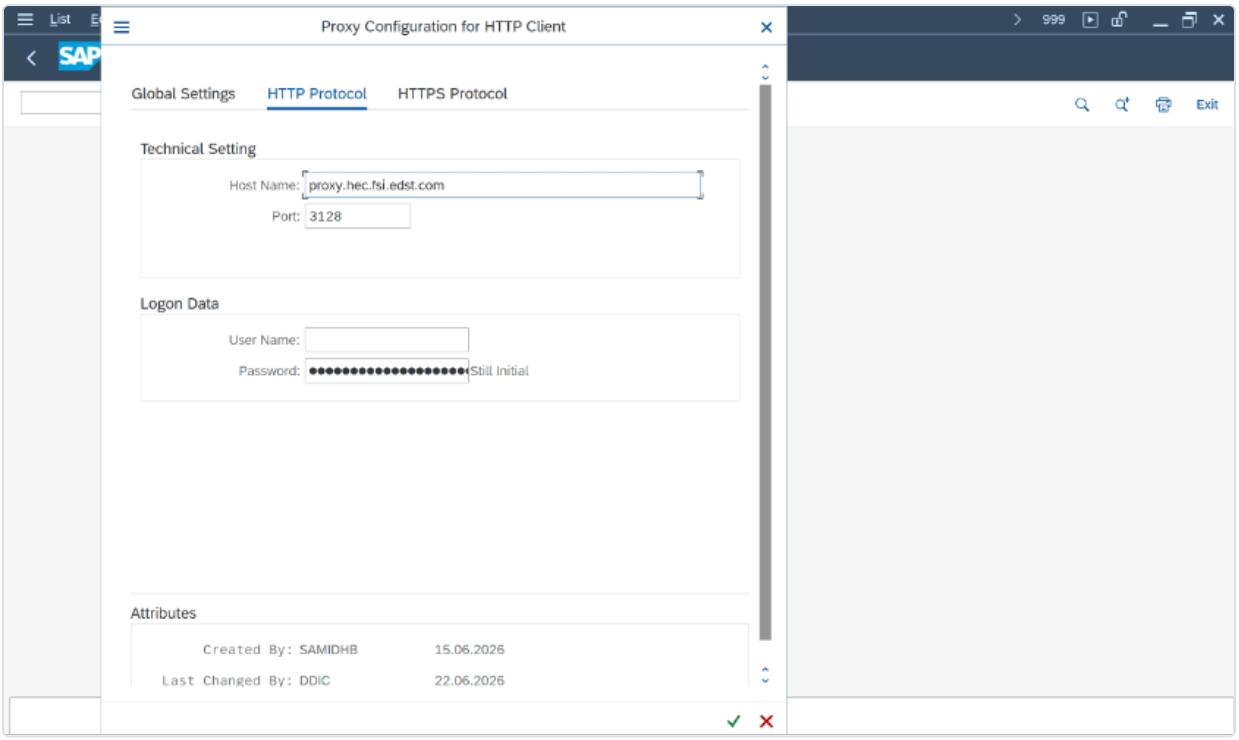
Response text for Connection to IBM GITHUB

SAP proxy settings (SICF)

Where the SAP system connects to GitHub through an outbound proxy, configure the corresponding proxy settings in transaction SICF, as shown in the following figures.



SAP proxy configuration, global settings



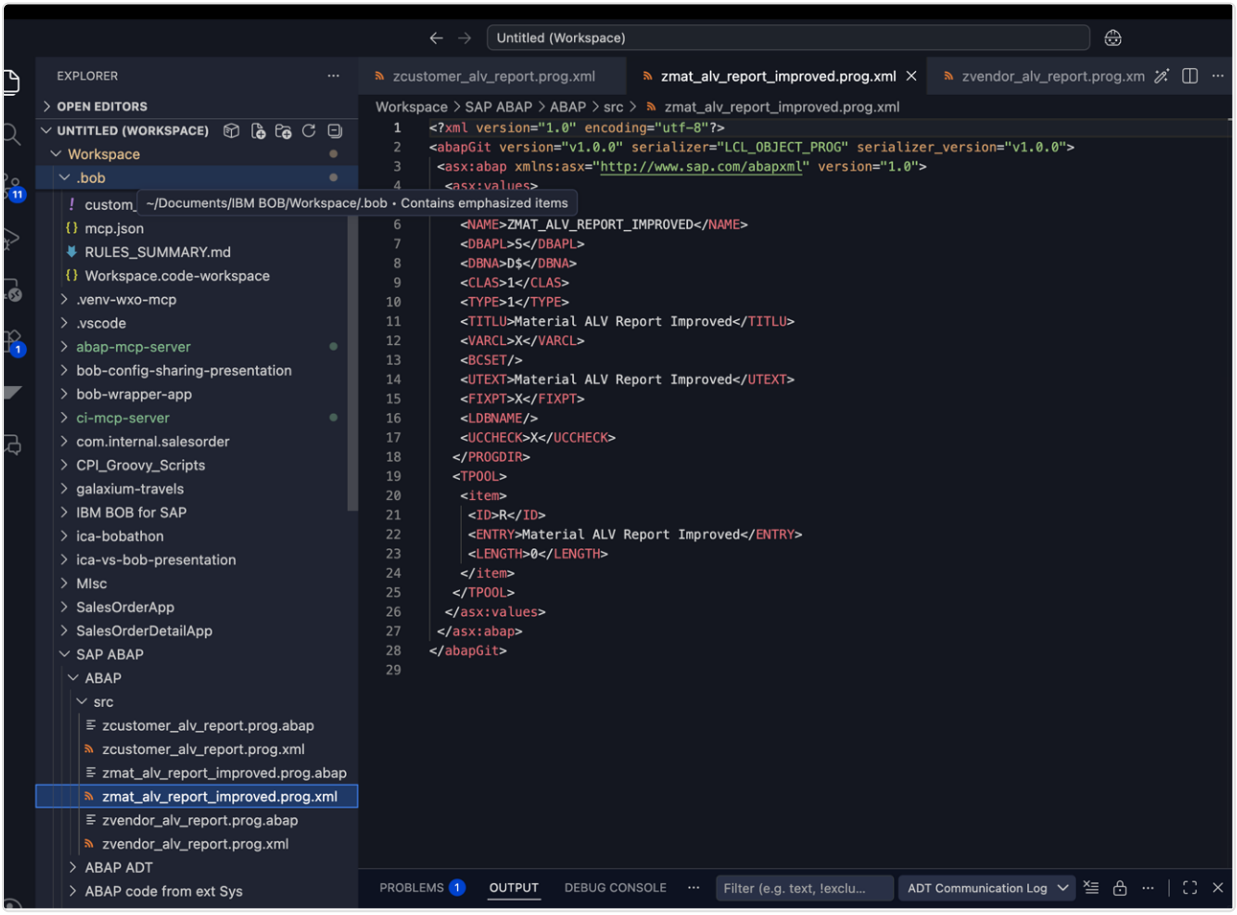
SAP proxy configuration, HTTP protocol settings

Walkthrough: Creating new ABAP code with IBM Bob and deploying to SAP

This walkthrough demonstrates the complete pipeline for a newly created ABAP program, from initial generation in IBM Bob through activation in the SAP back end.

Generate the ABAP program in IBM Bob

Log in to IBM Bob and create a new SAP ABAP program using an appropriate prompt, together with the relevant skill and rule configuration.



IBM Bob workspace with the generated ABAP program and its metadata file

In this example, IBM Bob generated the program REPORT ZMAT_ALV_REPORT_IMPROVED, together with its associated metadata file, zmat_alv_report_improved.prog.xml.

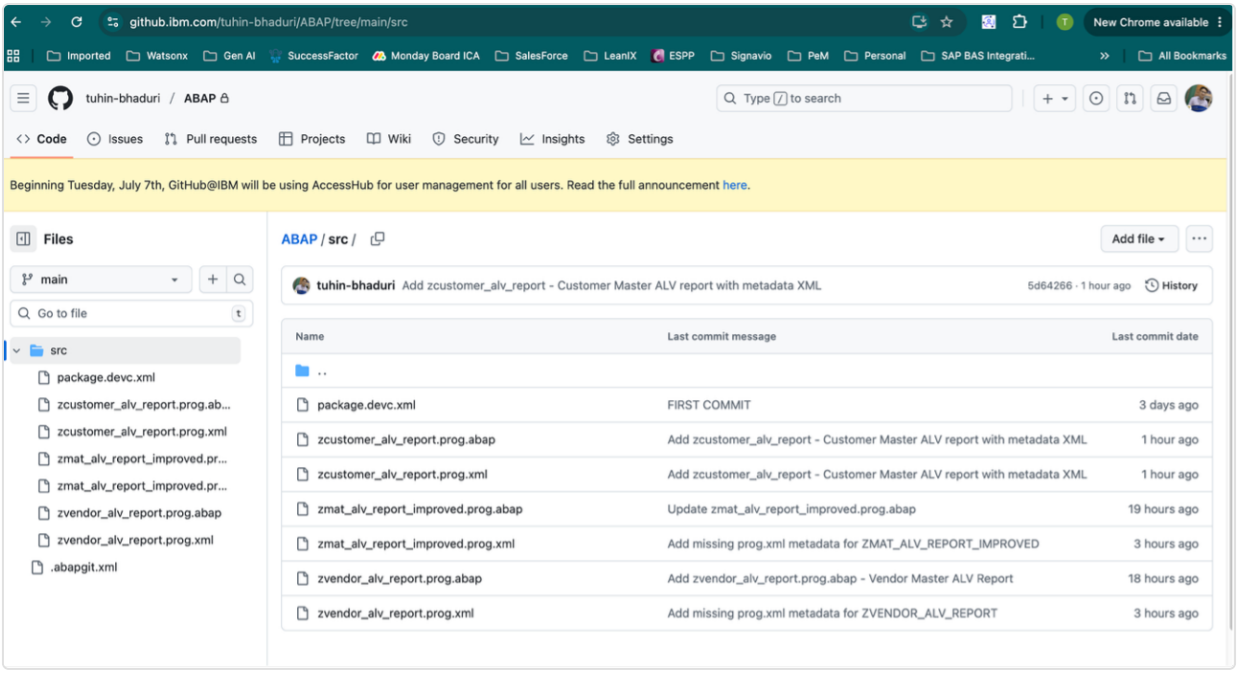
> **Note:** The metadata .xml file is essential for deployment through abapGit. If this file is missing, the program cannot be created in the SAP back end. Always confirm that the metadata file has been generated alongside the source file.

Push the new program to the Git repository

The generated ABAP program is now pushed to the Git repository — in this example, https://github.ibm.com/tuhin-bhaduri/ABAP, using the folder path ABAP/src/. This can be performed with standard CLI commands, or by prompting IBM Bob directly, for example:

Sample Prompt: “Move @SAP\ABAP/ABAP/src/ABAP_Mat_Improved.prog.abap and the associated metadata .xml file to the IBM GitHub repository at https://github.ibm.com/tuhin-bhaduri/ABAP.git.”

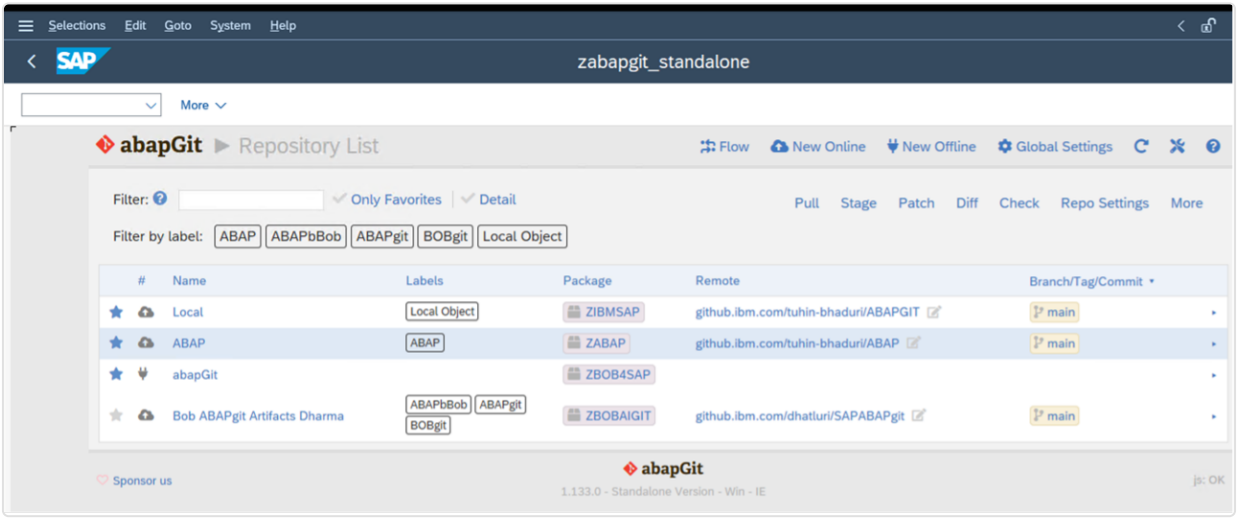
IBM Bob runs the required steps and pushes the changes to the target repository. Refreshing the repository confirms that the new files are present, as shown in the following figure.



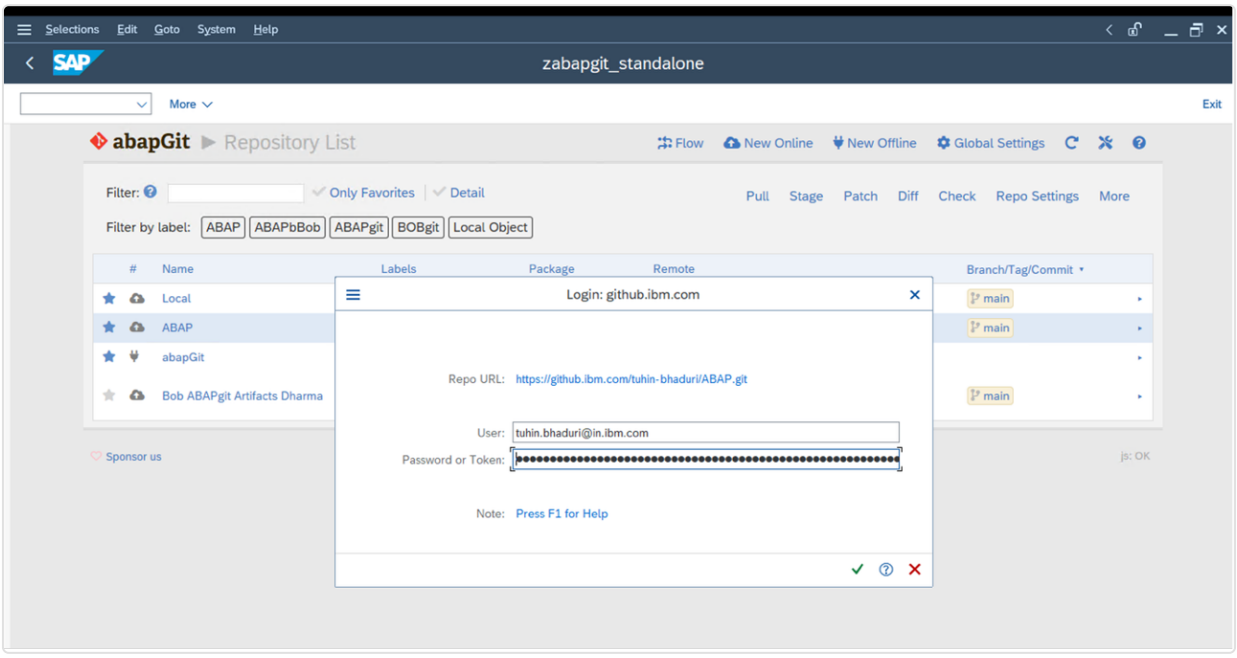
IBM GitHub src folder after pushing the new ABAP programs

Pull the changes into SAP through abapGit

With the changes now available in the Git repository, they must be pulled into the SAP system through abapGit. Log in to the SAP back end and start transaction ZABAPGIT.

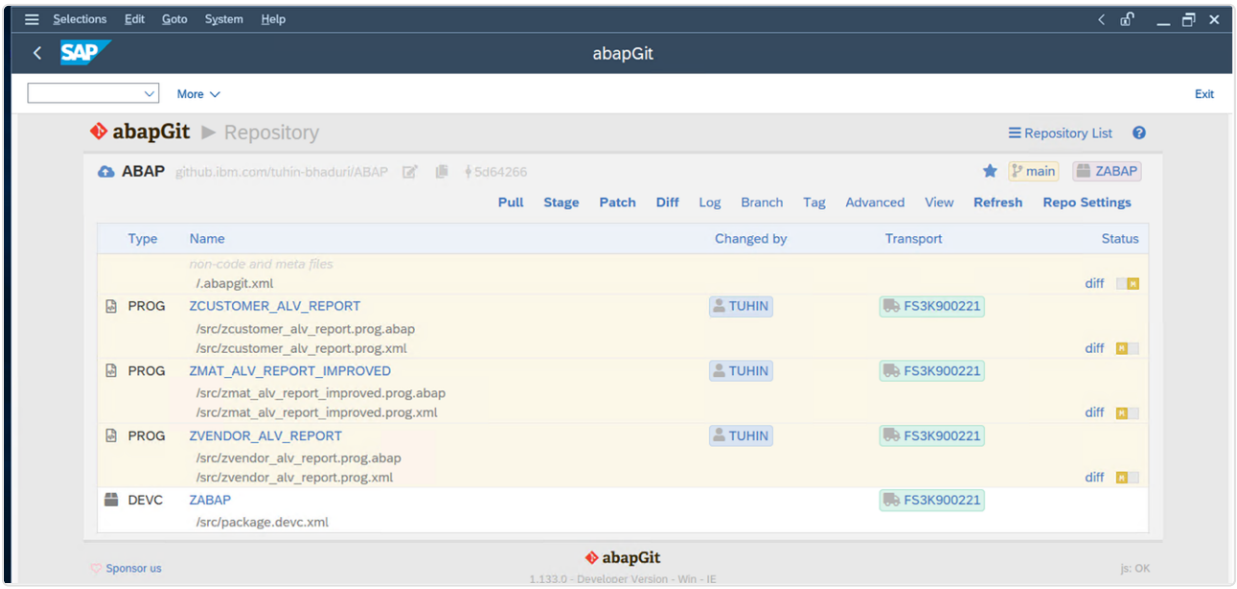


abapGit repository list with the ABAP repository ready to pull



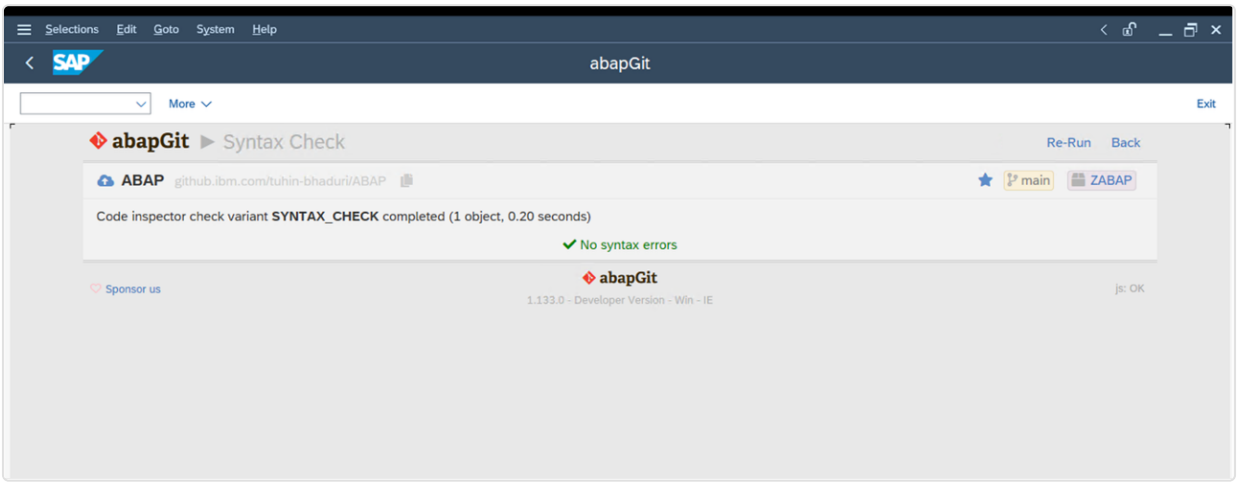
abapGit login prompt for github.ibm.com

After providing the token created for the IBM GitHub connection, the repository opens. Select Pull to retrieve all content from the Git repository into abapGit.



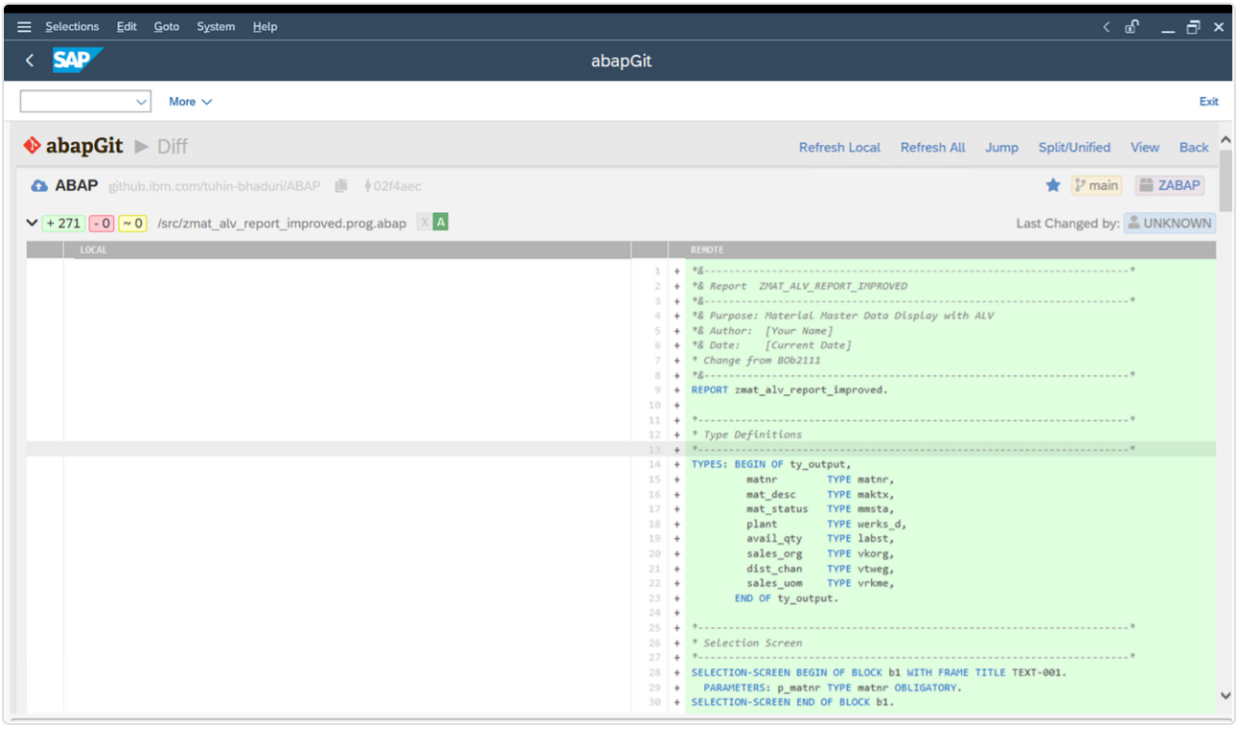
abapGit repository view with objects pending pull

The new code is now available in the abapGit online repository and can be validated using standard SAP quality checks — syntax check, Code Inspector, and unit testing. After these checks are satisfactory, the object can be assigned to a package and transport request.



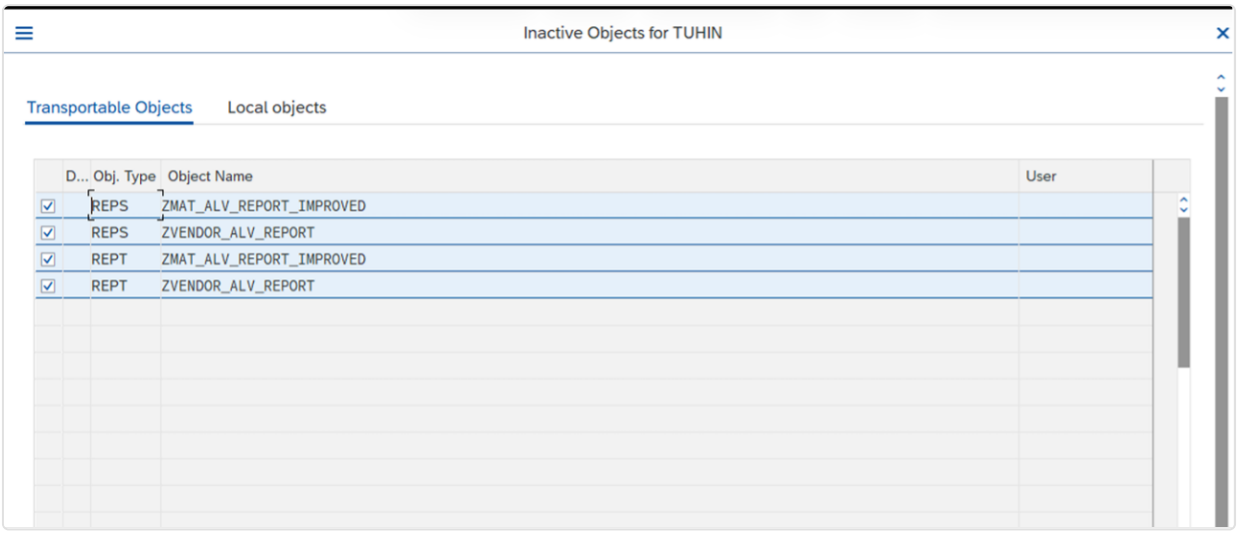
abapGit syntax check with no errors

The differences between the local and remote versions can be reviewed at any point before activation, as shown in the following figure.

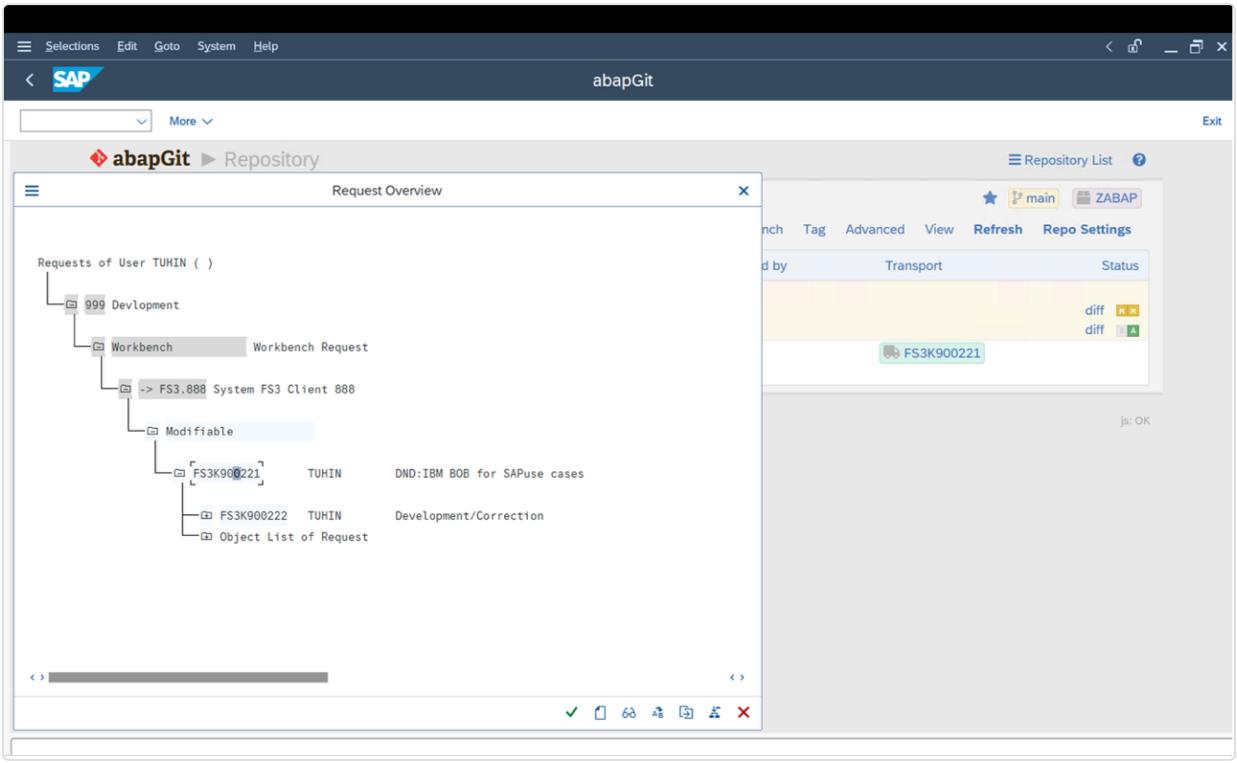


abapGit diff of ZMAT_ALV_REPORT_IMPROVED with 271 added lines

After the review is complete, the change is pulled from the abapGit explorer, which prompts for activation of the inactive object into a package and transport request.

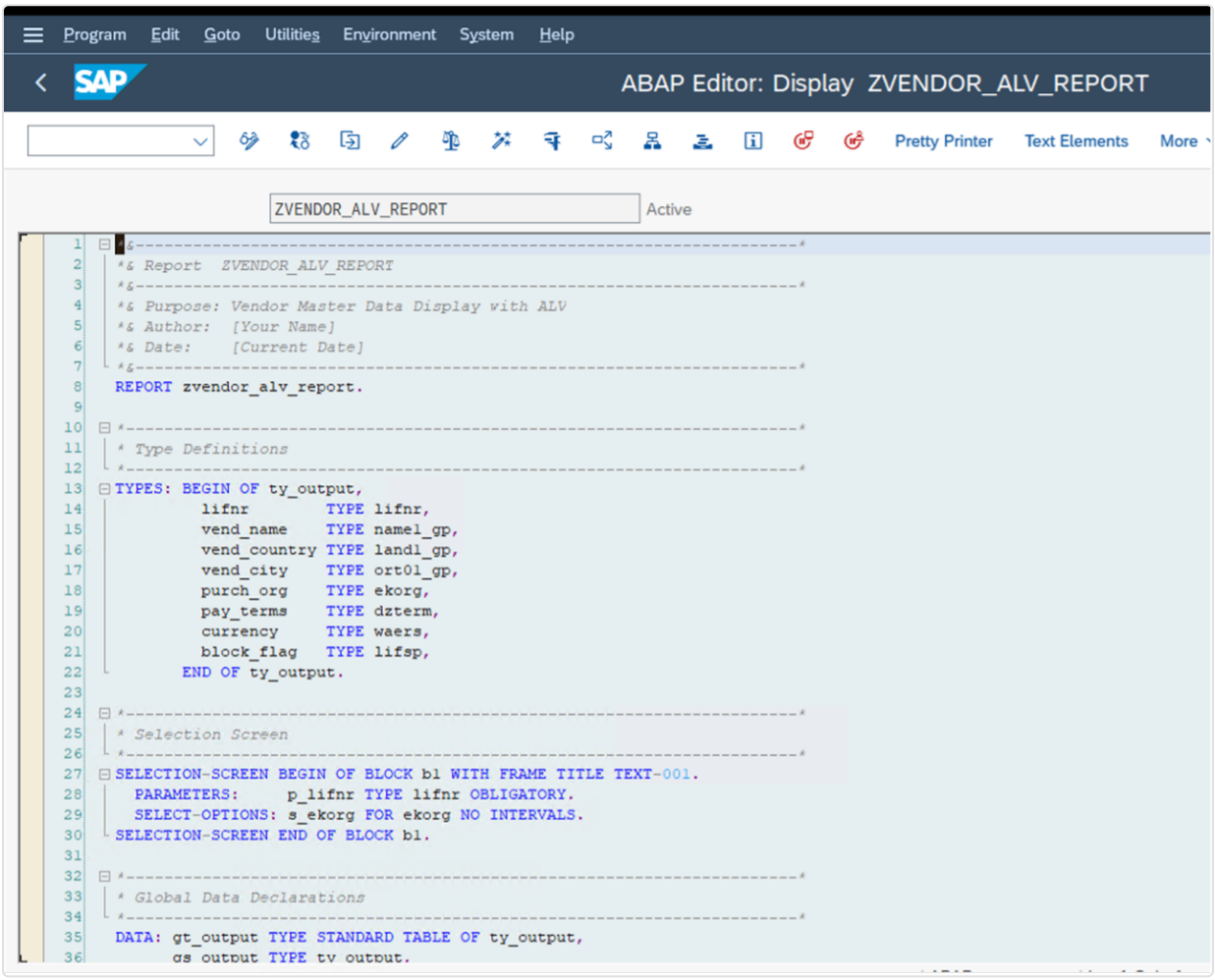


SAP Inactive Objects dialog listing objects to activate

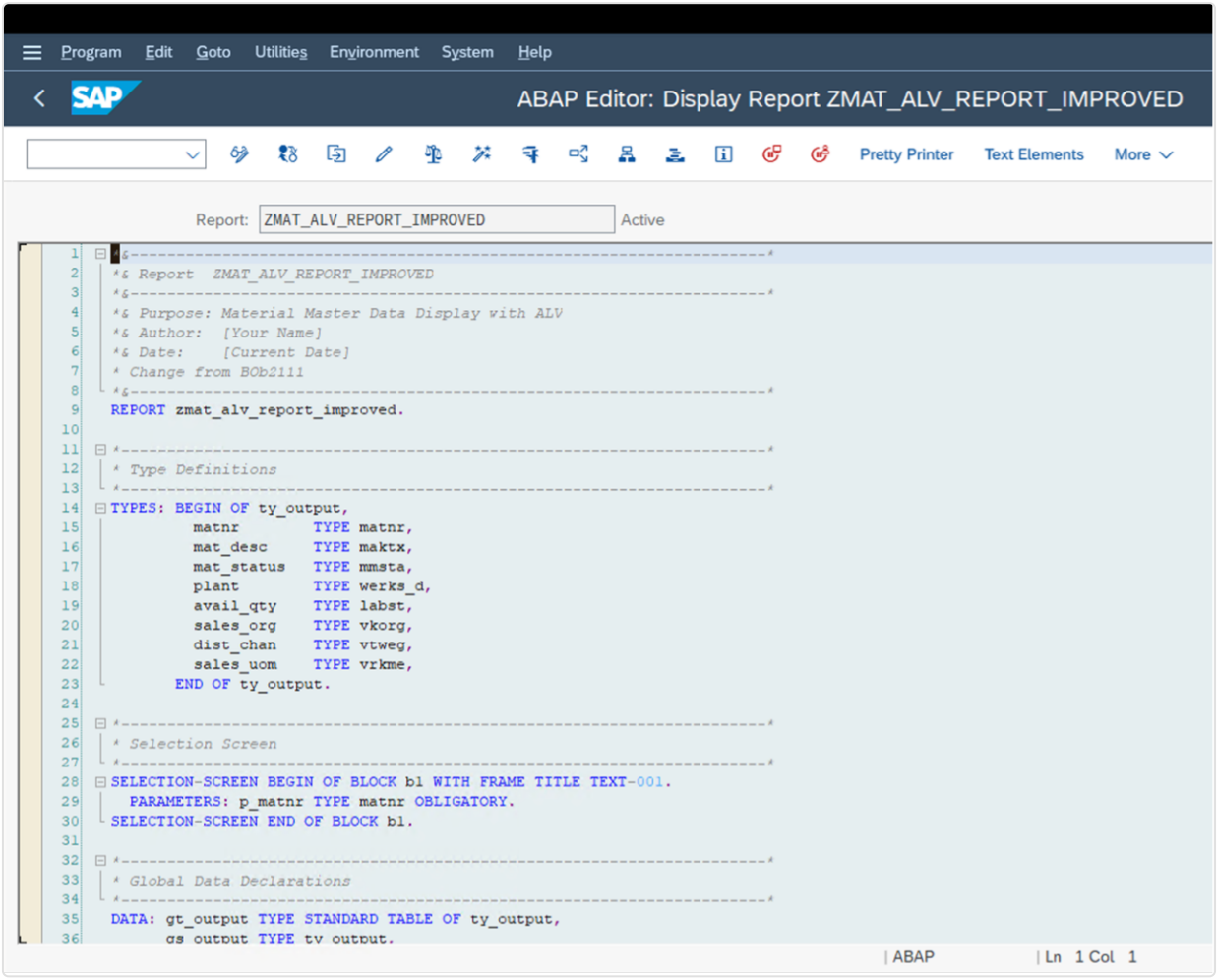


abapGit Request Overview with available transport requests

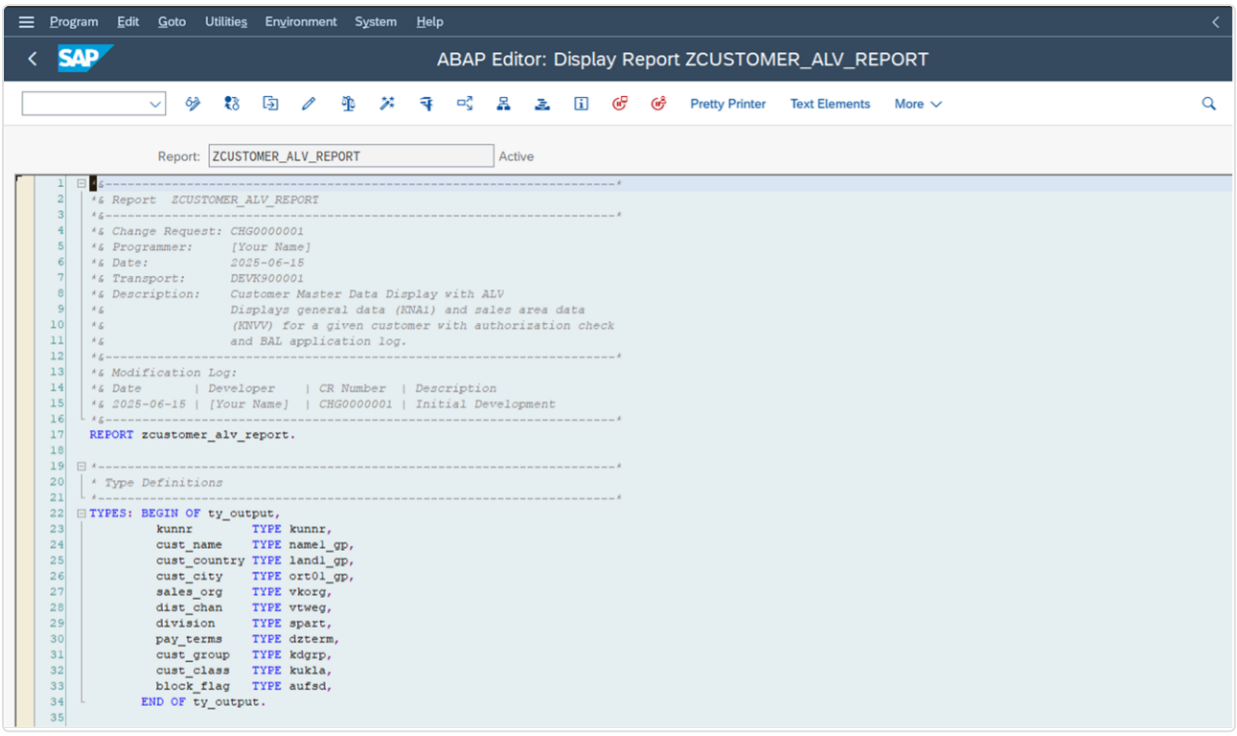
The activated program can then be confirmed directly in transaction SE38, as shown in the following figures.



ABAP Editor showing the activated program ZVENDOR_ALV_REPORT



ABAP Editor showing the activated program ZMAT_ALV_REPORT_IMPROVED



ABAP Editor showing the activated program ZCUSTOMER_ALV_REPORT

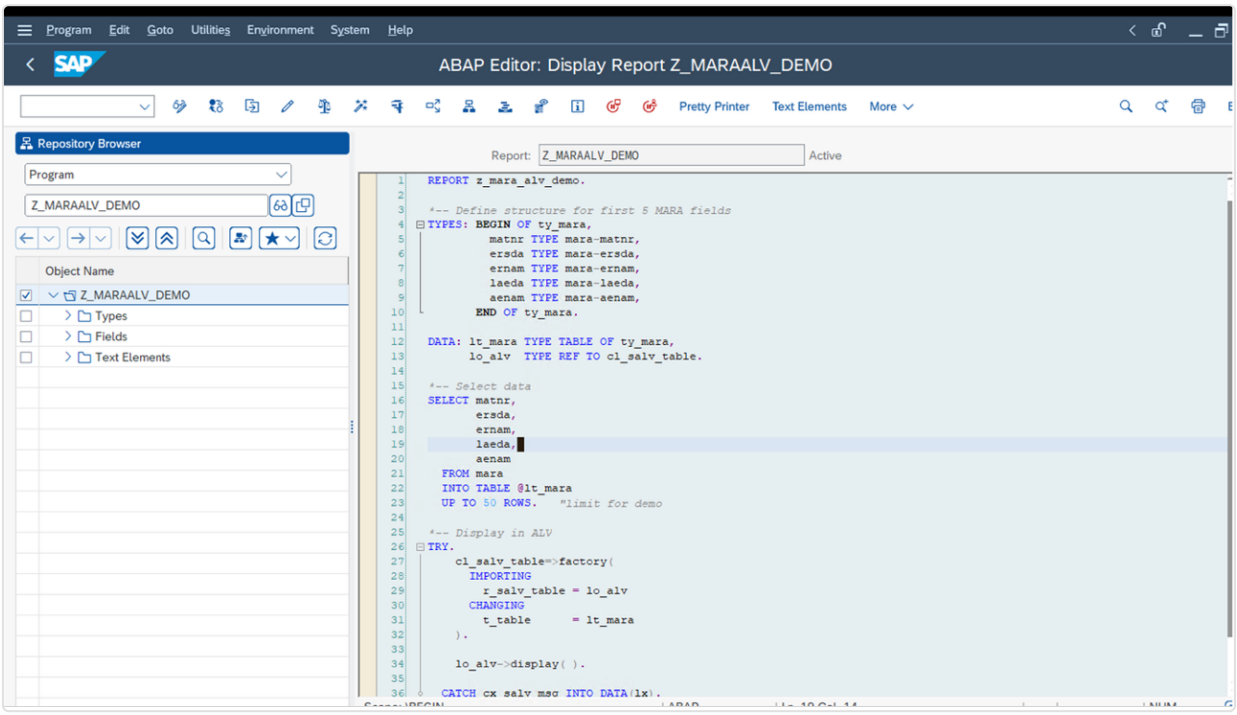
> **Attention:** The abapGit tool used in this section is the open-source community version of abapGit (available at https://github.com/abapGit/abapGit), owned and maintained by the SAP community. For support or to report issues, see the corresponding community resources at https://docs.abapgit.org/development-guide/. The abapGit documentation is made available under the Creative Commons Attribution 4.0 International license.

Walkthrough: Modifying existing ABAP code with IBM Bob and deploying to SAP

This walkthrough describes the corresponding process for enhancing or correcting an existing ABAP program using IBM Bob.

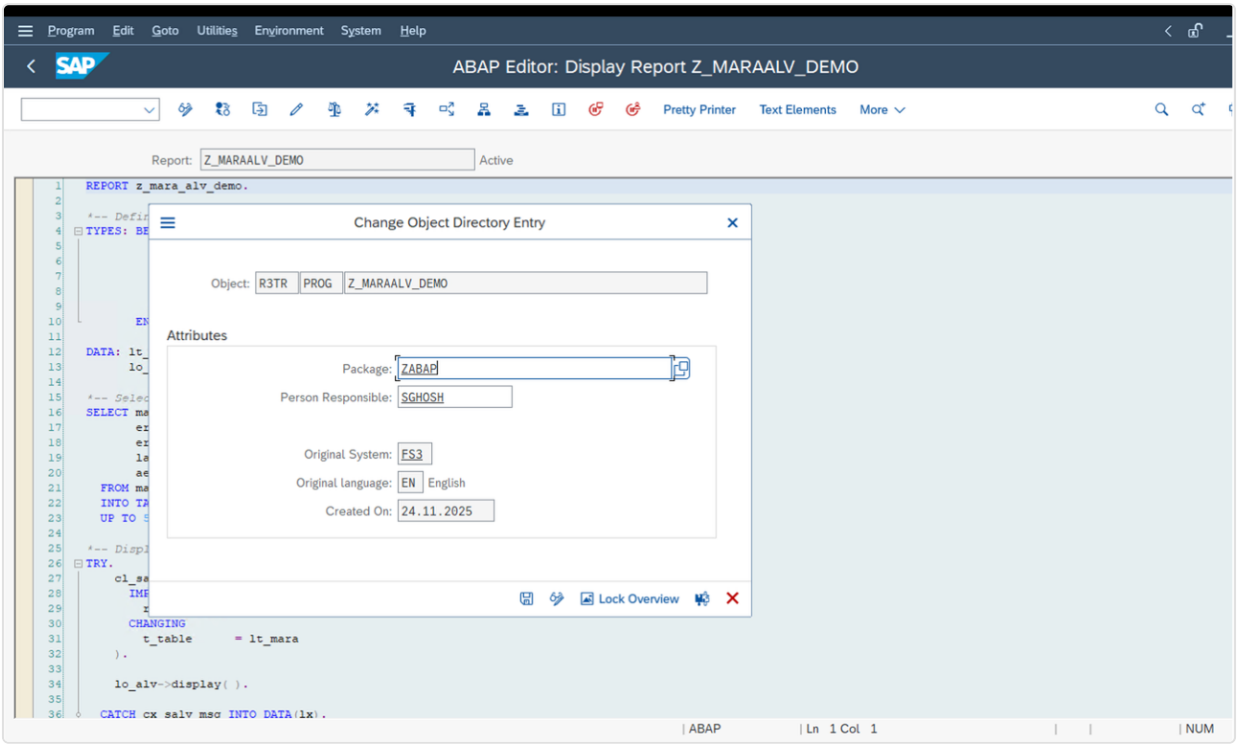
Identify the ABAP object to be enhanced

In this example, the existing program Z_MARAALV_DEMO is selected for enhancement using IBM Bob.



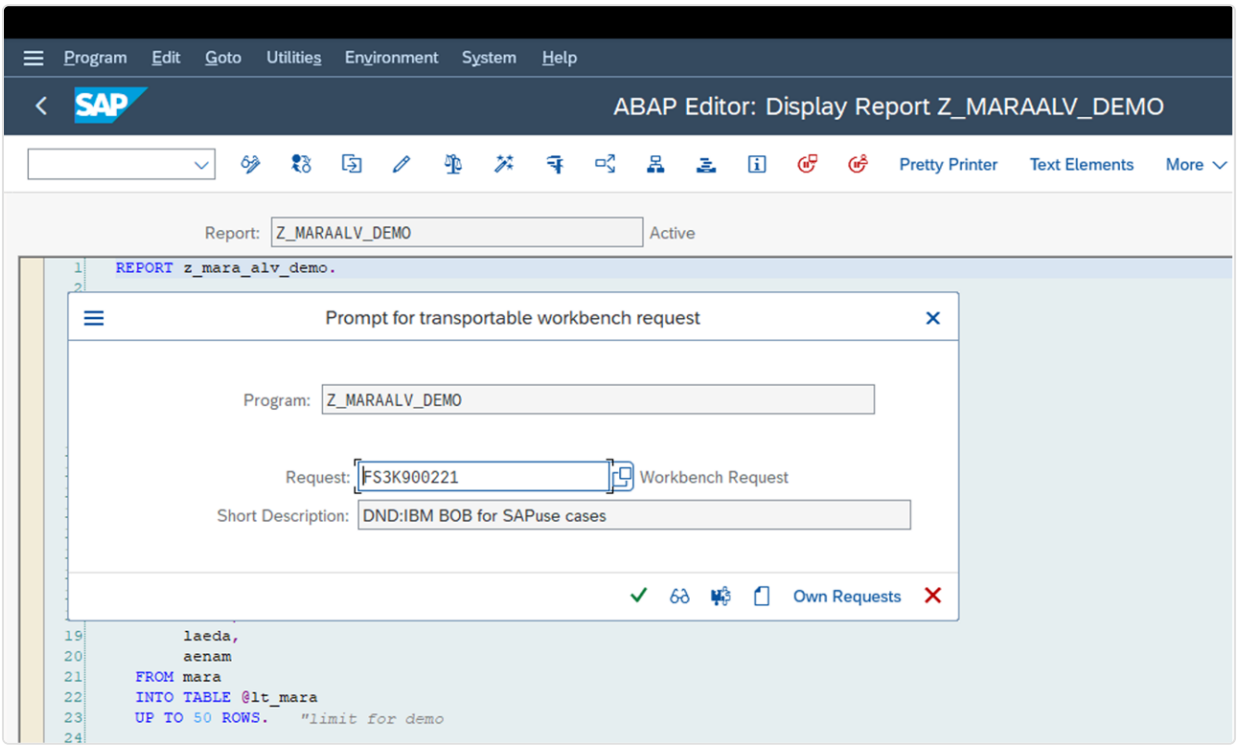
ABAP Editor showing the existing program Z_MARAALV_DEMO

Change the program's package assignment to the package associated with the abapGit online repository — in this example, ZABAP.



Change Object Directory Entry dialog for Z_MARAALV_DEMO

Review the current package. Reassign the program to package ZABAP.

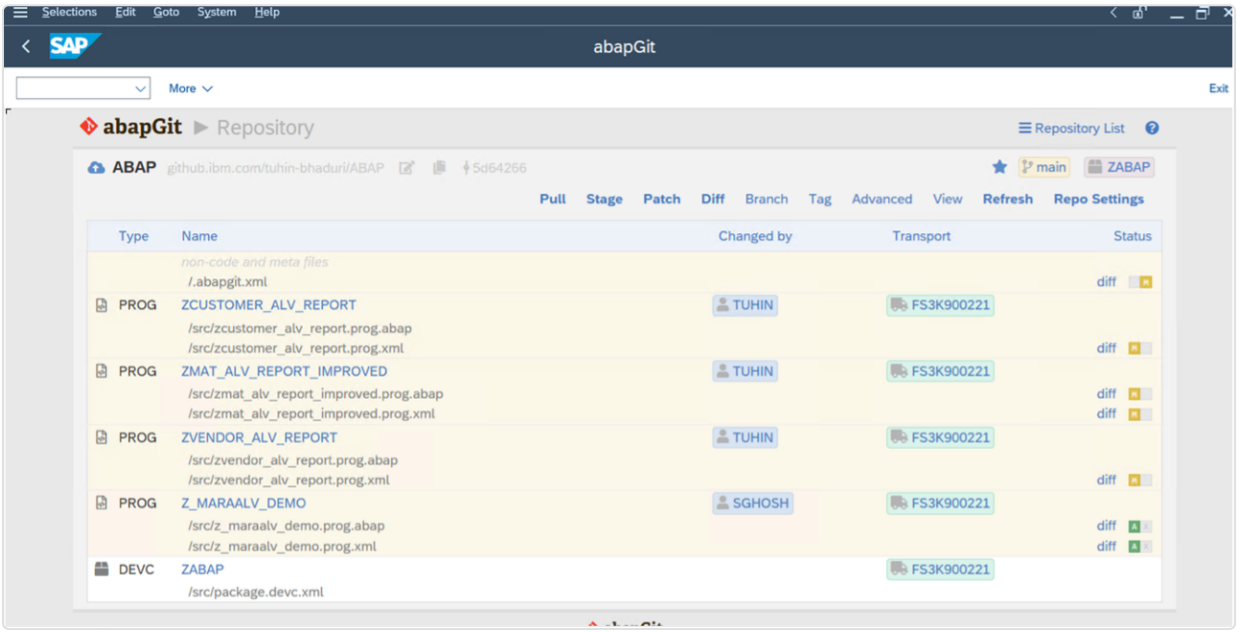


Transport request prompt for Z_MARAALV_DEMO

After this is complete, select the appropriate transport request.

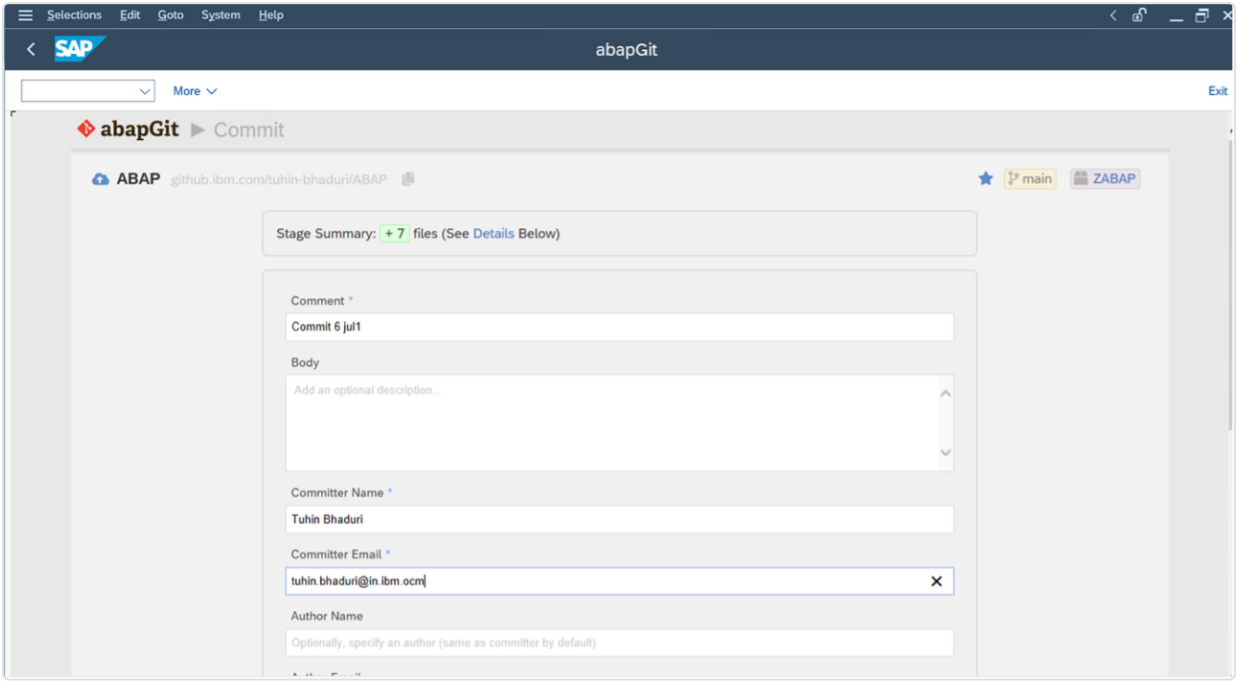
Push the existing program to the Git repository

With the package reassigned, the program now appears in abapGit. Push the program from abapGit to the Git repository, in this example https://github.ibm.com/tuhin-bhaduri/ABAP.



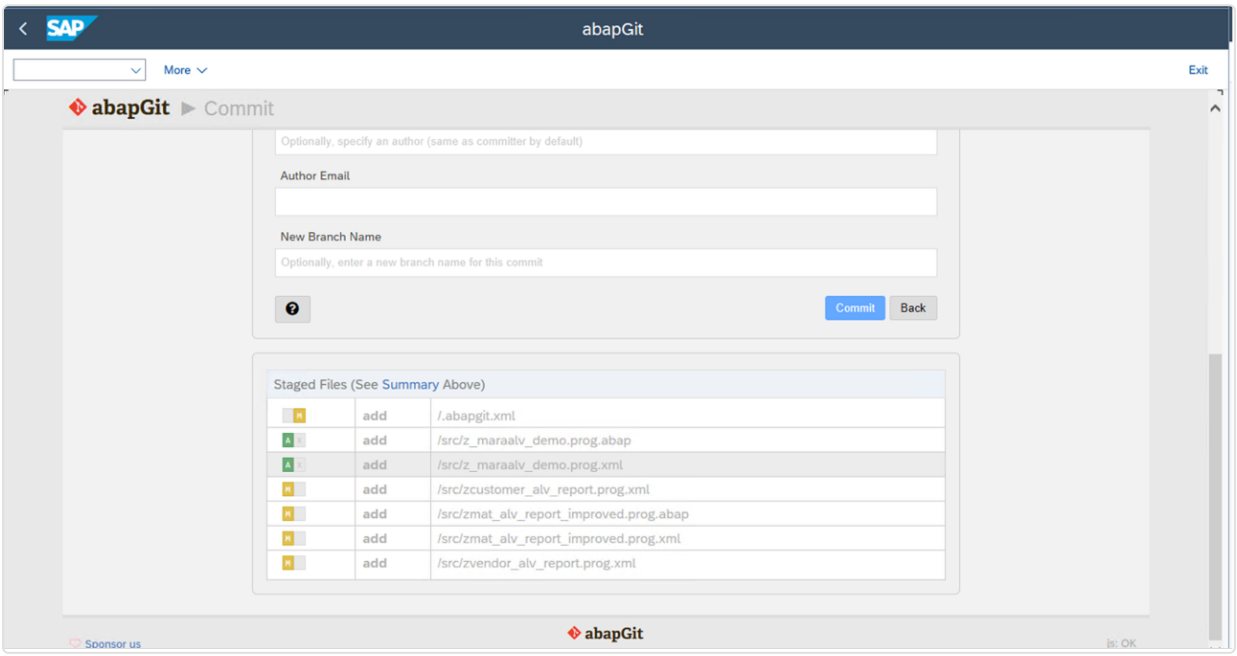
abapGit repository view with Z_MARAALV_DEMO added

Following the stage-and-commit operation, both the metadata file and the ABAP source file are available in the Git repository.



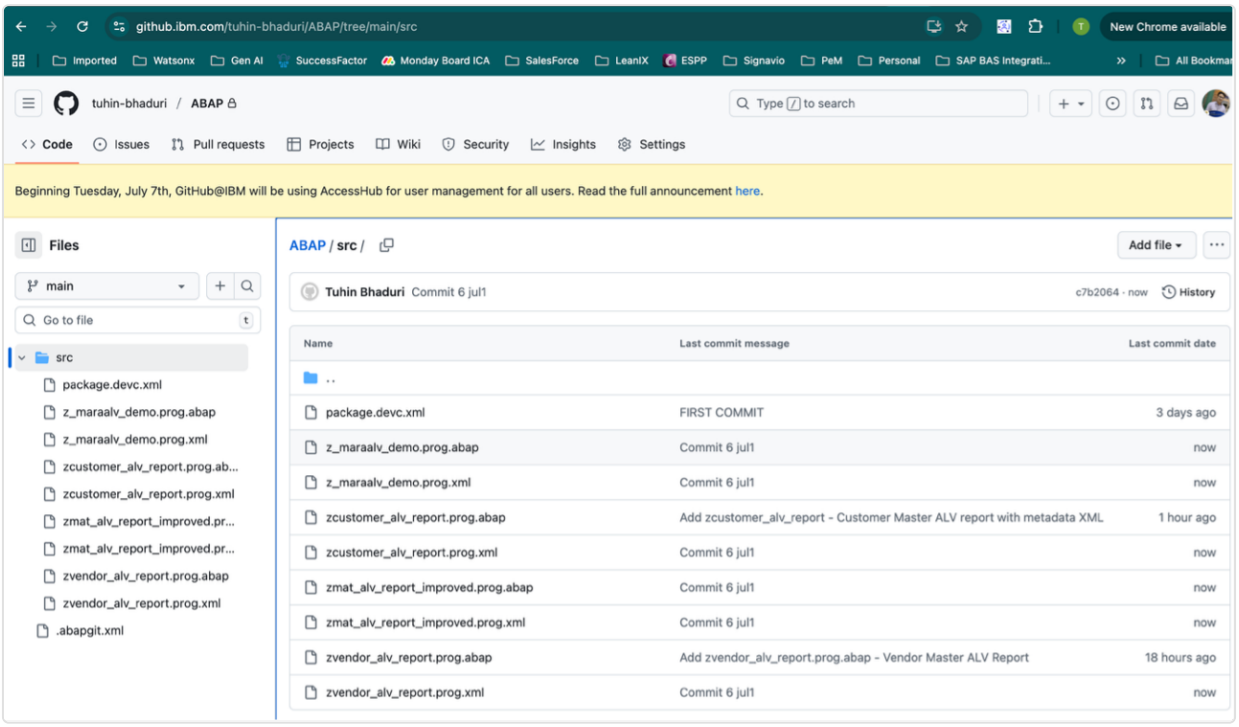
abapGit commit screen with seven staged files

The metadata and source files are now available in the Git repository.



abapGit commit screen with the staged files list

Review the commit history for the pushed files.



IBM GitHub src folder after the commit

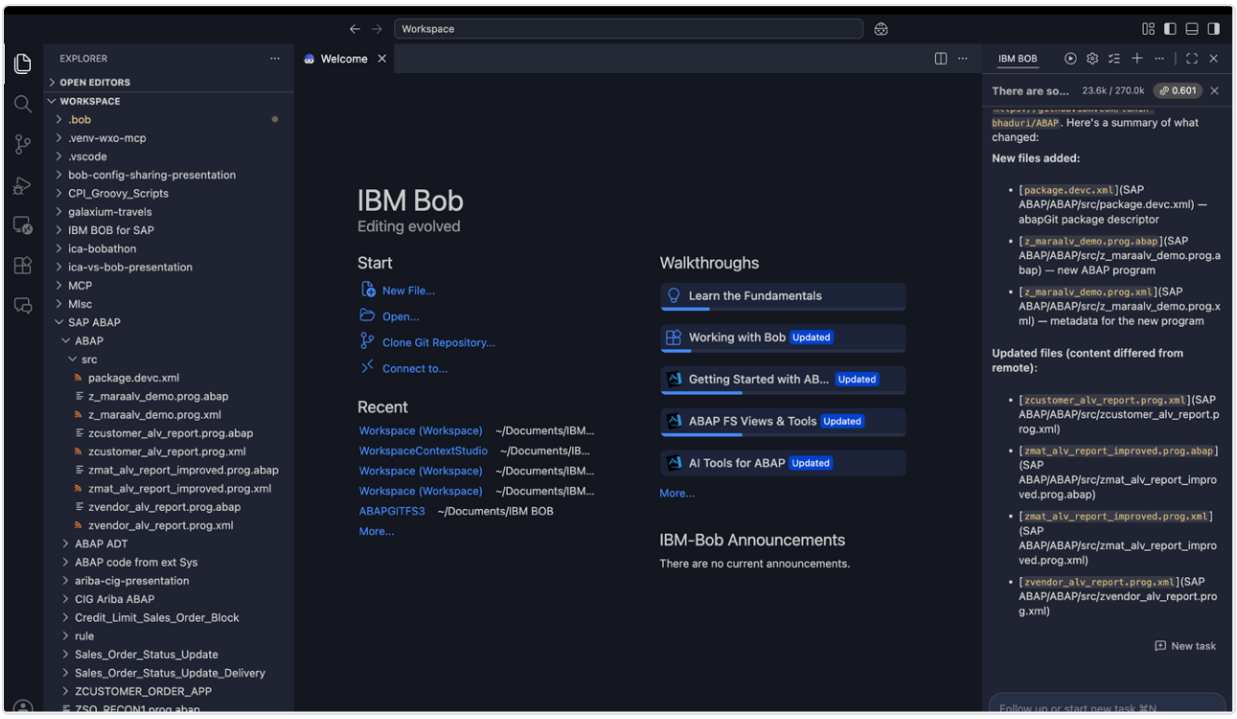
The file listing confirms that both artifacts are present.

Clone the repository into IBM Bob

Return to IBM Bob and clone the repository into the workspace so that IBM Bob has access to the existing program.

Sample Prompt: “There are some files added in https://github.ibm.com/tuhin-bhaduri/ABAP. Please update the folder SAP ABAP/ABAP/src based on the same Git repository.”

IBM Bob creates, or clones, the repository at the specified location.



Cloning the Git repository into the IBM Bob workspace.

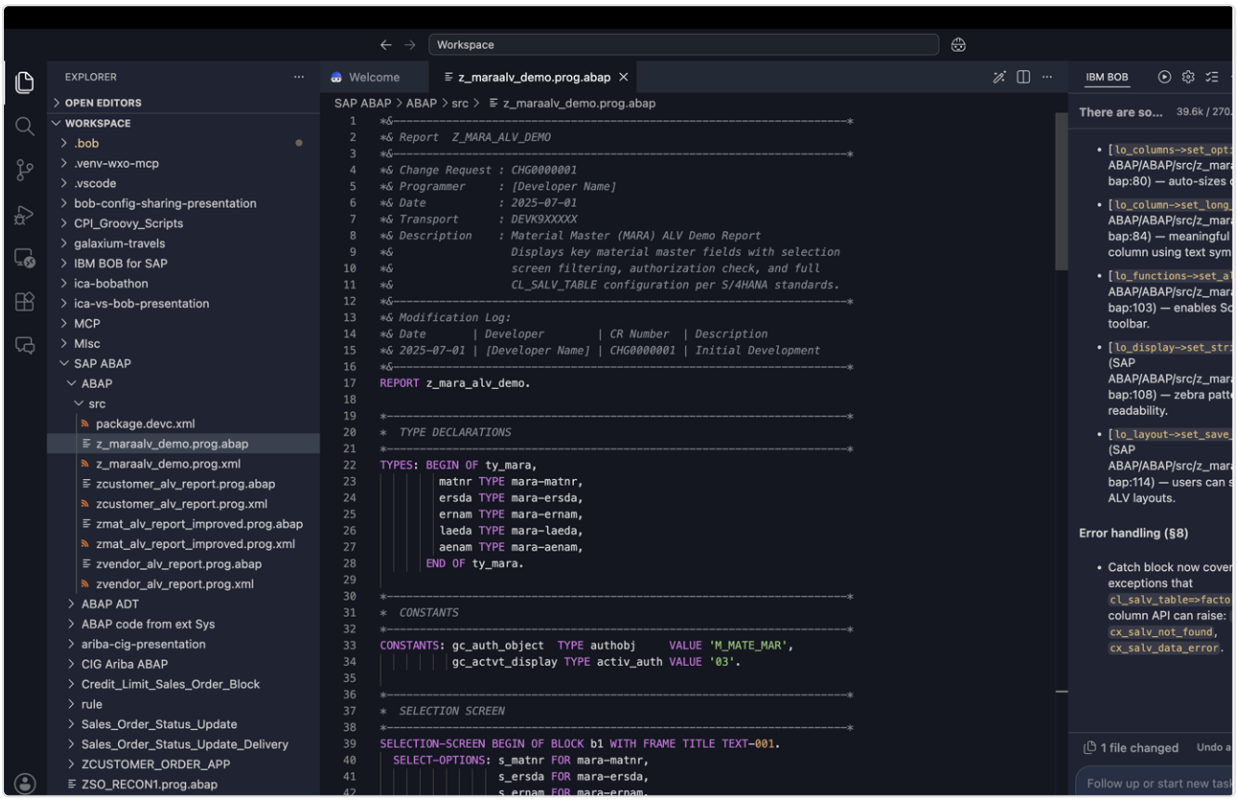
Download and save the code, and confirm that Z_MARAALV_DEMO has been copied into the workspace.

Enhance the program with IBM Bob

Prompt IBM Bob to review and rewrite the program in line with the project's defined skills, rules, and ABAP best practices for SAP S/4HANA:

Sample Prompt: “Please review the code at @SAP\ABAP/ABAP/src/z_maraalv_demo.prog.abap. Apply the relevant skill and rules, and rewrite the program following ABAP best practices for SAP S/4HANA.”

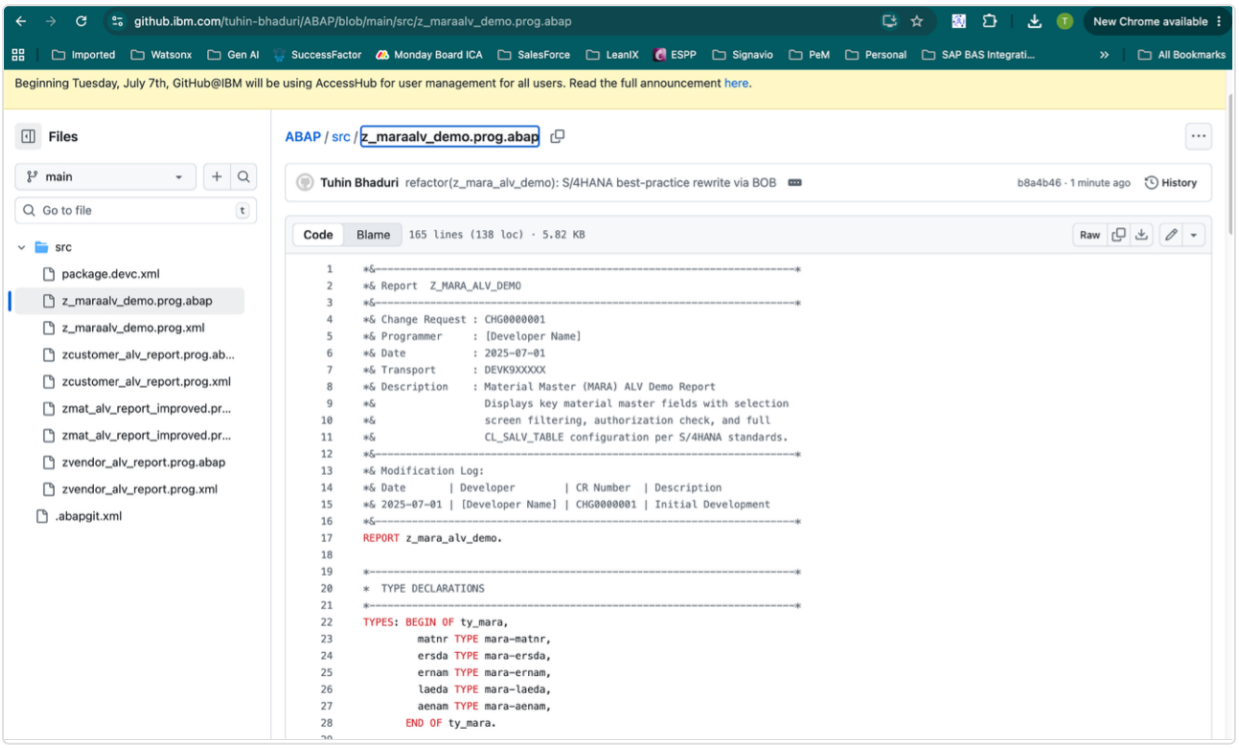
IBM Bob rewrites the program accordingly.



IBM Bob rewriting the program in line with defined skills and rules.

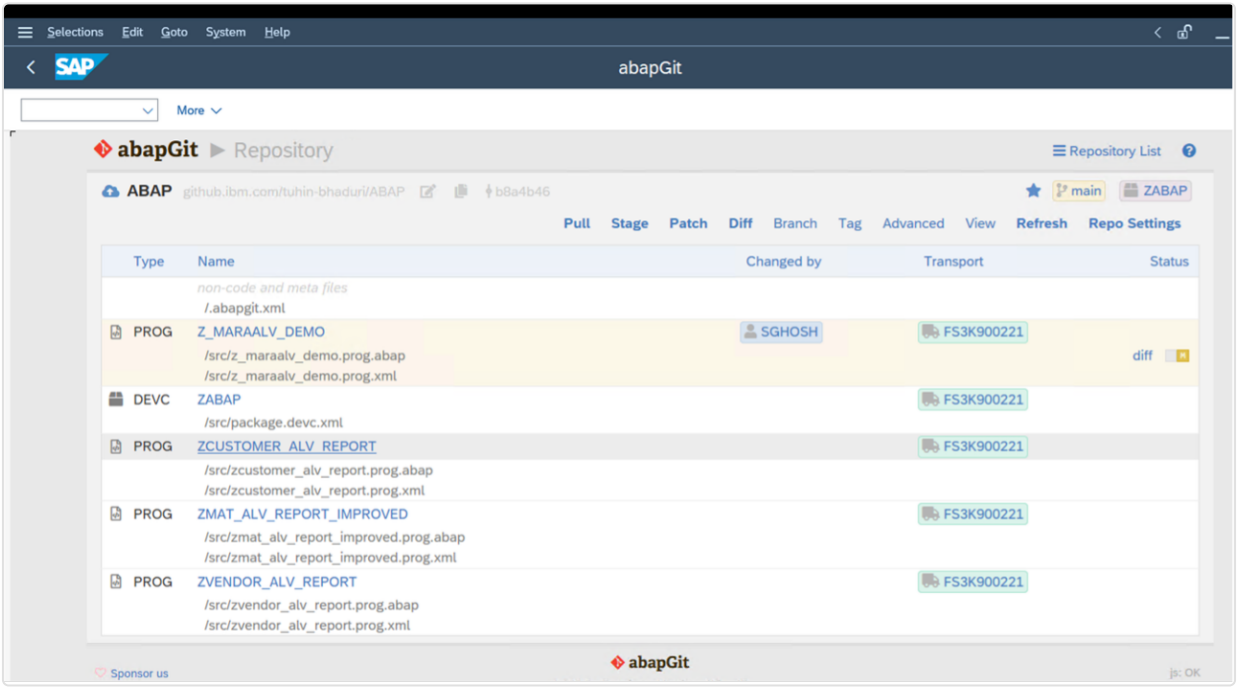
Push the enhanced program and deploy to SAP

Push the updated program from IBM Bob to the Git repository, for example with the prompt “Push the changes from Bob to IBM Git.” The revised program authored by IBM Bob is now available in the IBM Git repository.

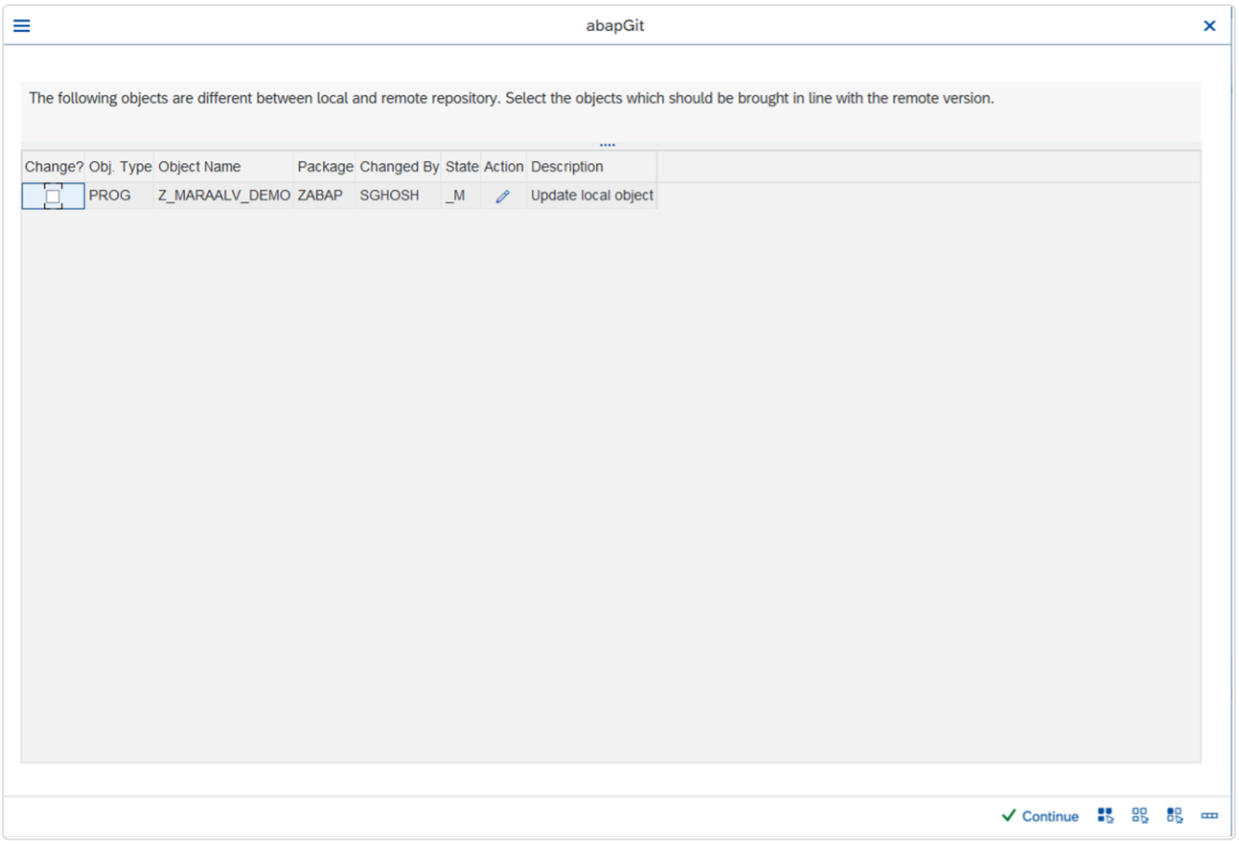


Enhanced program pushed to the IBM Git repository.

In the SAP back end, open abapGit and the corresponding online repository, and select Pull.

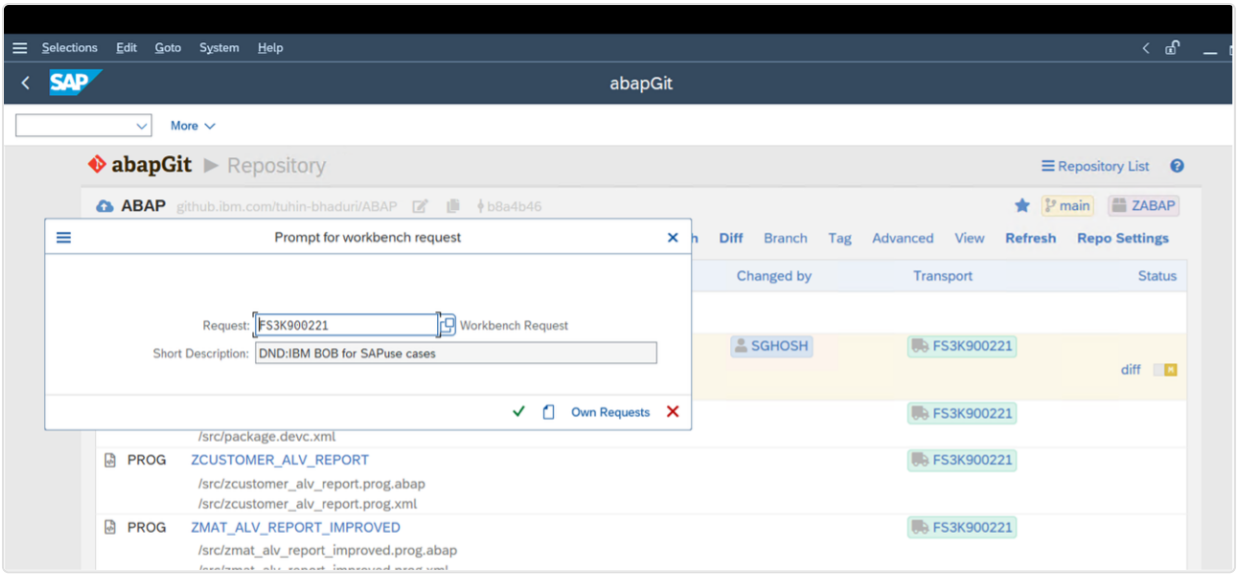


Pulling the enhanced program into abapGit.

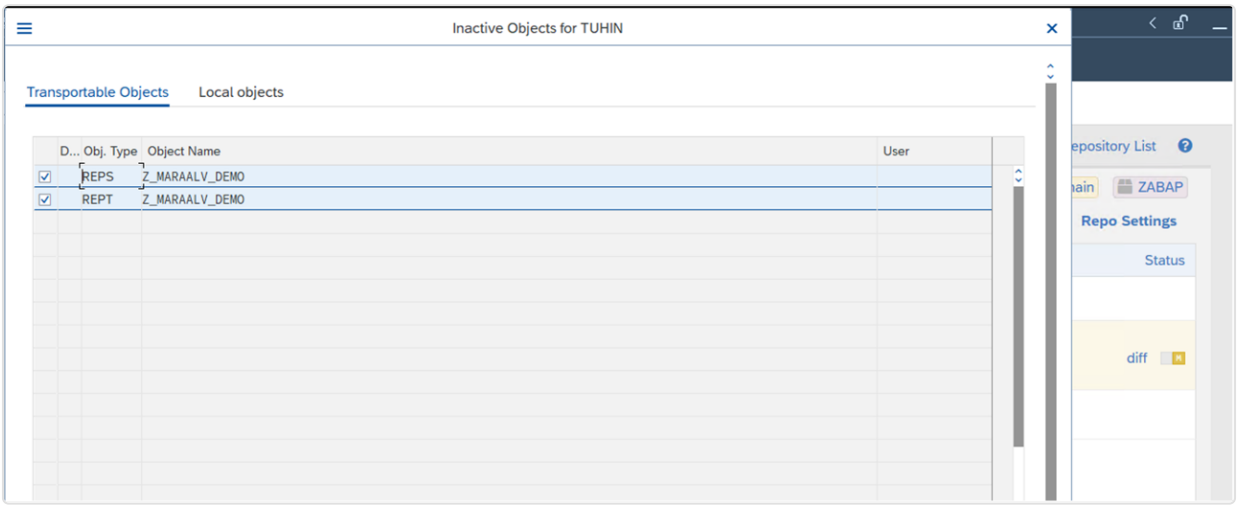


Reviewing the pulled changes prior to activation.

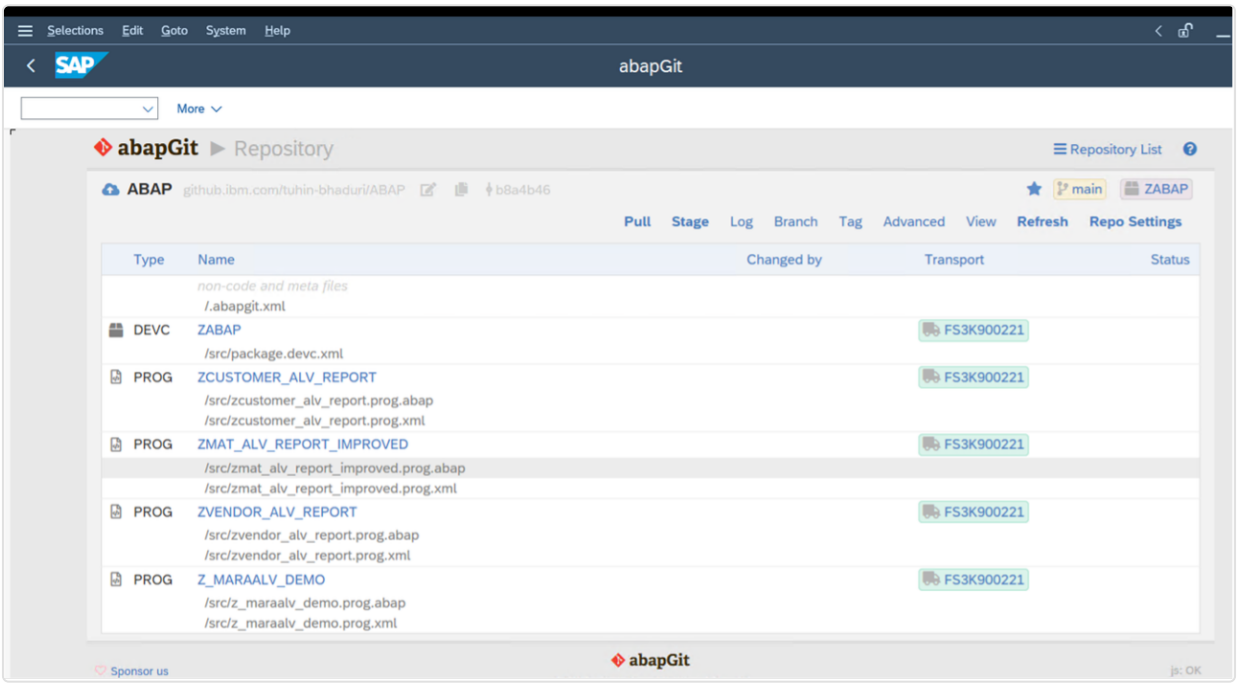
After selecting the relevant objects and choosing Continue, the change proceeds through the standard activation flow.



Confirming object selection before activation.

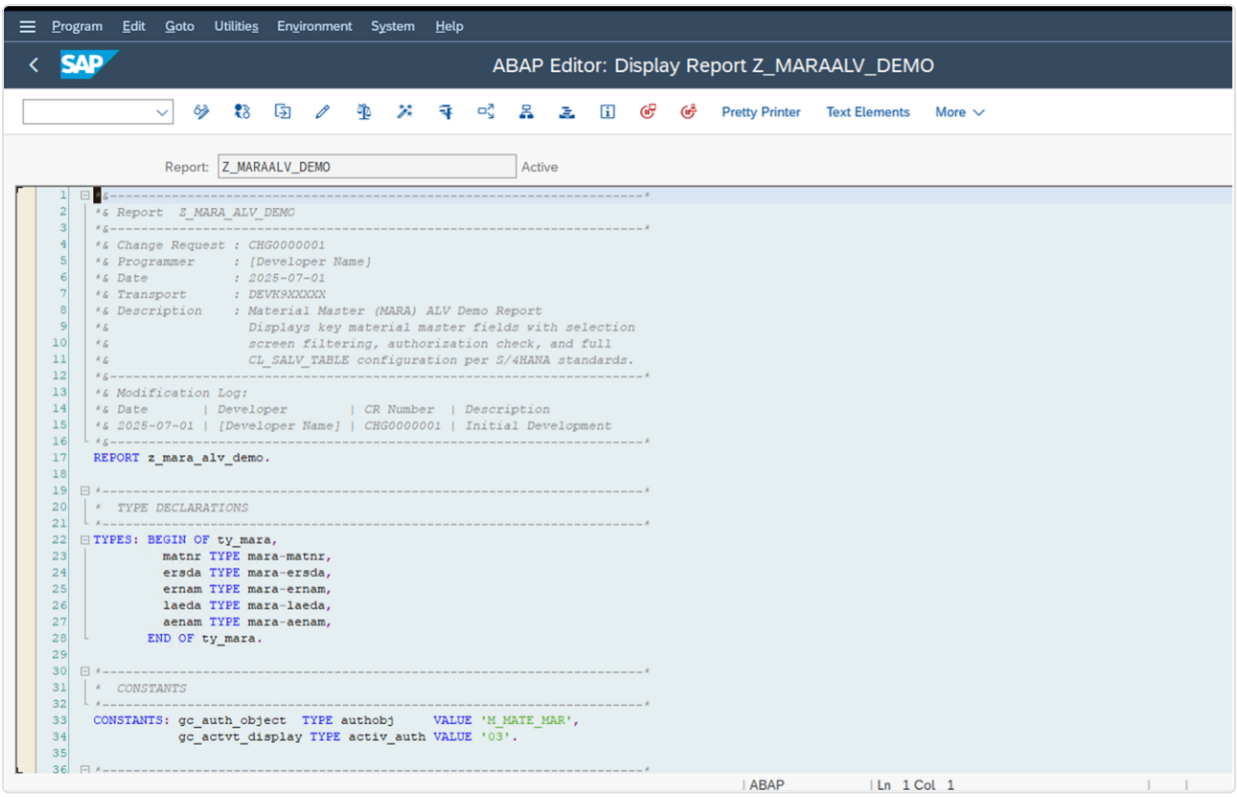


Activation confirmation.



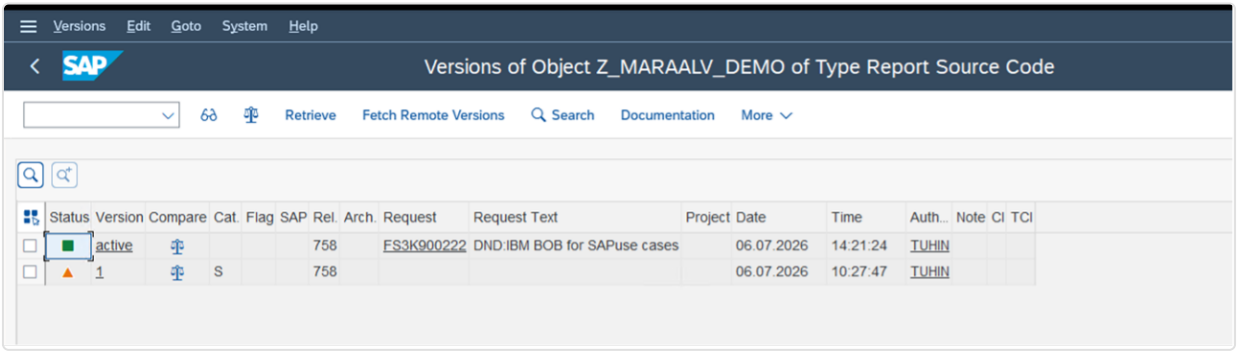
Final activation status.

The updated logic can then be confirmed directly in transaction SE38.



Verifying the enhanced program in SE38.

The program's version history confirms that the change originated from the IBM Git repository through IBM Bob, providing a complete, auditable record of the change.



Version history confirming the change originated from IBM Bob through IBM Git.

Governance and business value

The architecture described in this section delivers value across every stakeholder group involved in SAP development:

- Developers gain a single, familiar Git-based workflow that applies equally to code written by hand and code generated or refactored with AI assistance, eliminating ambiguity about which version of an object is current.
- Architects gain a documented, repeatable reference architecture for integrating AI-assisted development tools with SAP without weakening existing controls over object activation and transport.
- Project managers gain improved predictability, because concurrent changes are surfaced and reconciled in Git before they reach the SAP system, reducing the risk of late-cycle transport conflicts and deployment failures.

- Senior management gains a complete, permanent audit trail — every change, author, and timestamp recorded in Git — supporting compliance, quality, and change-management objectives as AI-assisted development scales across the organization.

Disclaimer and licensing

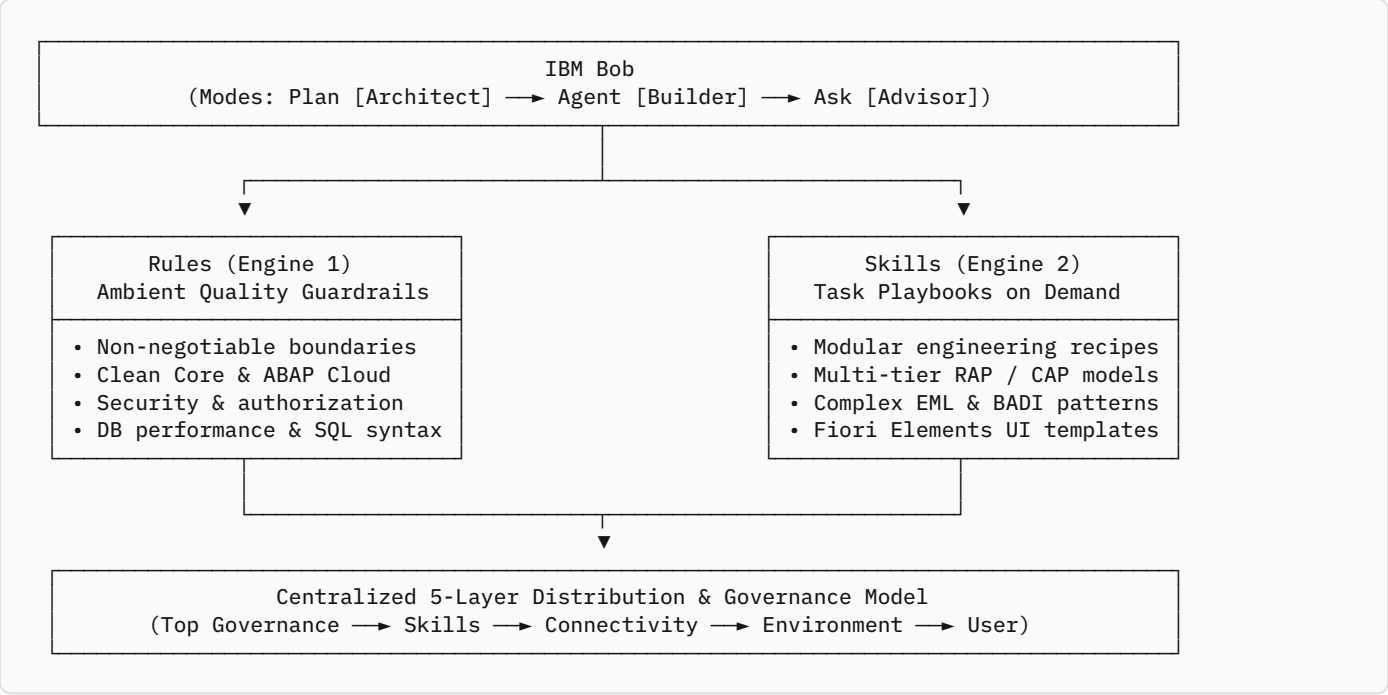
abapGit, as used throughout this section, is the open-source community edition of abapGit, available at https://github.com/abapGit/abapGit. The software is owned and maintained by the SAP community. For issues or support requests, consult the corresponding community resources at https://docs.abapgit.org/development-guide/. The abapGit documentation referenced in this section is distributed under the Creative Commons Attribution 4.0 International license.

For more information, see the abapGit section of the Additional Resources.

Working with IBM Bob

Adopting AI in mission-critical enterprise environments like SAP requires far more than casual code autocompletion. Organizations must ensure that AI output strictly adheres to non-negotiable architectural mandates—such as SAP Clean Core, ABAP Cloud language scopes, ATC quality gates, and stringent corporate naming and security standards—while empowering development teams to deliver complex multi-tier applications at speed.

This section establishes the core operational framework of **IBM Bob**. It details how Bob orchestrates its built-in personas (Agent, Plan, and Ask) alongside domain-specific Custom Modes, controls risk through a fine-grained two-level permission model, and enforces architectural compliance using a revolutionary **Dual-Engine Governance model** (Rules as persistent ambient guardrails and Skills as on-demand technical playbooks).



What this section covers

- **IBM Bob Operational Modes and Persona Lifecycle**: In-depth examination of built-in operational modes—**Agent Mode (The Builder)** for end-to-end execution, **Plan Mode (The Architect)** for structured requirements

analysis and architectural blueprints, and **Ask Mode (The Advisor)** for risk-free code exploration—along with specialized **Custom Modes** tailored to enterprise roles.

- **Two-Level Permission Model and Risk Management**: Comprehensive breakdown of tool-group permissions (`read`, `edit`, `execute`, `mcp`, `skill`, `workflow`, `todo`, `subtask`, `subagent`, `mode`) and auto-approval policies to protect critical environments and preserve human-in-the-loop governance.
- **Dual-Engine Governance (Rules vs. Skills)**:
  - **Rules (The Project Constitution)**: Ambient, continuously active guardrails (`.bob/rules/`) enforcing SAP Clean Core principles, ABAP Cloud syntax, released API usage (`I_*` CDS views), strict SQL optimization, and mandatory security checks.
  - **Skills (The Expert Playbooks)**: Dynamically activated domain instruction sets (`.bob/skills/`) that guide multi-artifact choreography across RAP (CDS, BDEF, behavior pools, SRVD, SRVB), CAP Node.js services, and Fiori Elements.
- **Enterprise Standards and Naming Conventions**: Practical rule templates across ABAP, UI5, database performance, authorization checks, unit testing, and transport readiness.
- **The 5-Layer Centralized Configuration and Distribution Model**: Architecture for distributing corporate Bob configurations (governance policies, skills, MCP connections, and environment variables) across global enterprise teams via managed Git repositories.
- **Spec-Driven Development and Knowledge Graph Integration**: Bridging the four key development gaps (Context, Specification, Validation, and Traceability) to transform high-level functional requirements into validated, production-ready enterprise code.

Overview of working with IBM Bob

For information about the ways to connect IBM Bob to an SAP system, see Connecting IBM Bob to SAP. Now the focus shifts: having connected IBM Bob, what does working with it actually look like day-to-day?

IBM Bob is not a single-mode tool. It is designed to adapt to the nature of the task at hand — sometimes acting as a hands-on developer, sometimes as a careful planner, sometimes as a knowledgeable advisor. Understanding how those modes work, and when to use each one, is the foundation of working with IBM Bob effectively.

IBM Bob modes: Three personas, one tool

IBM Bob ships with three built-in modes of operation — Agent, Plan, and Ask — plus the ability to define custom modes for specialized workflows. Each mode gives Bob a different persona, a different set of tools, and a different behavioral focus.



Figure: IBM Bob modes

Agent Mode

> **When to use it:** implementing features, fixing bugs, refactoring code, creating new objects, running builds or tests, or any task that requires actual changes to be made.

When a developer asks IBM Bob to implement a feature, fix a bug, or refactor a class, they want results — not a conversation about results. Agent mode is built for exactly this. Bob acts as an experienced software engineer, autonomously performing multi-step tasks: reading files, editing code, running commands, calling external services, and managing complex workflows end-to-end.

Agent mode is the only mode with Execute access — meaning it can run shell commands, build pipelines, and terminal operations. It carries the full toolkit: read, edit, execute, mcp, skill, workflow, todo, subtask, subagent, and mode switching. For ABAP development, this means Bob can read source objects through an MCP-connected SAP system, generate or refactor code, and guide the developer through activation and transport.

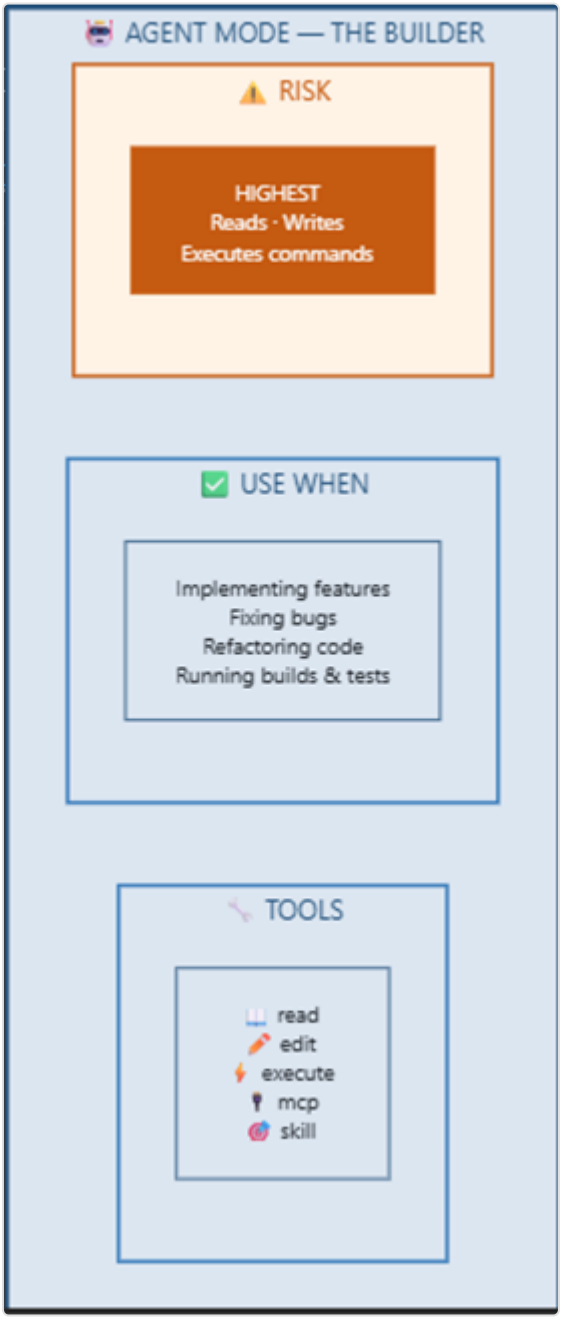


Figure: IBM Bob Agent mode — The Builder

Plan Mode

> **When to use it:** starting a new project or complex feature, breaking down a large problem, designing system architecture, or creating technical specifications before committing to implementation.

Before writing a single line of code, the best developers ask: do I understand the problem well enough to solve it correctly? Plan mode gives IBM Bob the space to ask that question seriously.

In Plan mode, Bob becomes a structured technical planner. Every session automatically loads the built-in create-plan skill, which guides Bob through requirements gathering, codebase analysis, and producing a step-by-step implementation plan for the developer to review and approve — before any changes are made.

Critically, Plan mode cannot run commands. It is intentionally kept in the thinking and design space. Once a plan is approved, Bob can switch autonomously to Agent mode to implement it — but not before.

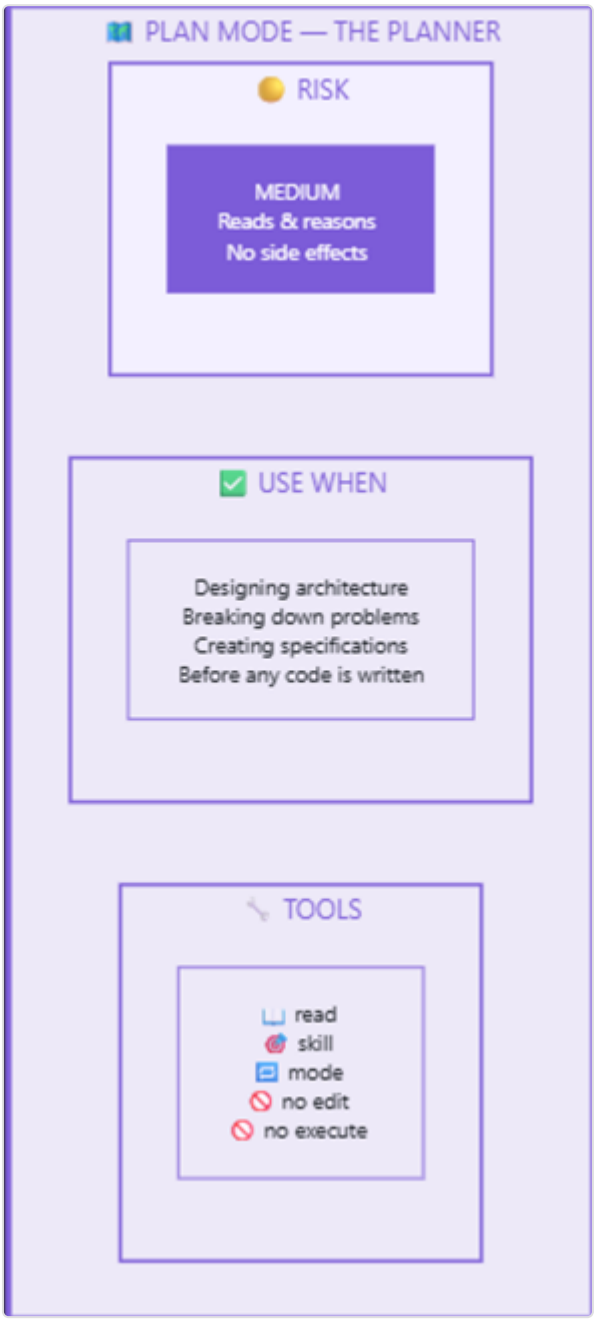


Figure: IBM Bob Plan mode — The Architect

Ask Mode

> **When to use it:** understanding unfamiliar code, learning about a technology or concept, getting architectural recommendations, exploring a codebase safely before making changes, or asking general technical questions.

Sometimes a developer does not need Bob to do anything. They need Bob to explain something — to help them understand an unfamiliar piece of code, think through an architectural decision, or explore a concept before committing to an approach.

Ask mode is read-only and conversational. Bob analyzes the codebase, explains how things work, provides recommendations, and includes Mermaid diagrams when they help clarify responses. It cannot write or modify files. It cannot run commands. It is the safest mode — ideal for exploration without risk.

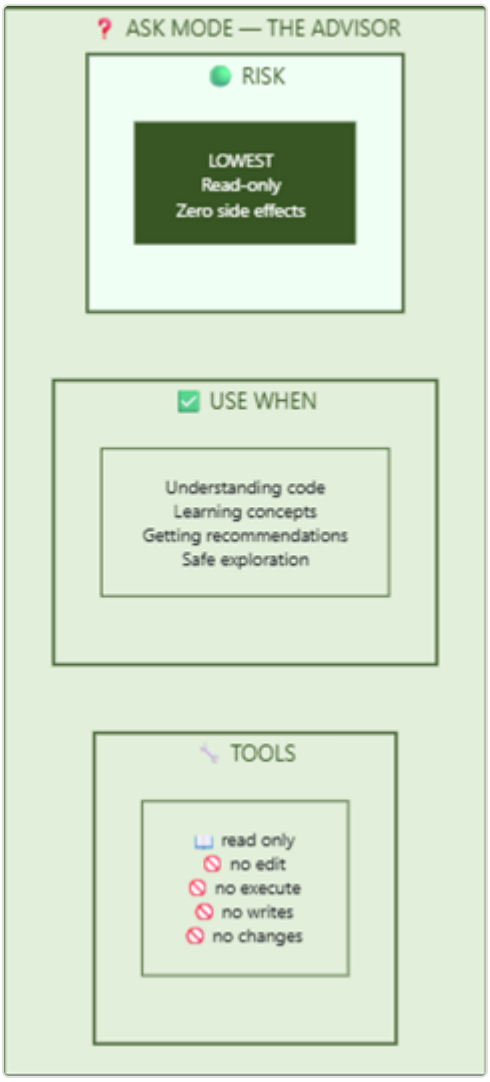


Figure: IBM Bob Ask mode — The Advisor

Custom Modes

Beyond the three built-in modes, IBM Bob supports custom modes — specialized personas defined by the organization for domain-specific workflows. A Security Reviewer mode that operates read-only and focuses on vulnerability analysis. A Production Operations Specialist that can query SAP through MCP but cannot modify code. A SAP ABAP Developer mode pre-configured with naming conventions, transport rules, and clean core standards already embedded.

Custom modes are defined in `.bob/custom_modes.yaml` at the project level, or `~/.bob/custom_modes.yaml` globally. Project-level modes override global ones, which override Bob's defaults. Once defined, they appear automatically as slash commands — a mode with slug security-reviewer becomes `/security-reviewer` instantly.

How to switch modes: Dropdown menu in the Bob panel | Keyboard shortcut `Ctrl+.` (Windows/Linux) or `Cmd+.` (macOS) | Slash commands `/agent`, `/plan`, `/ask` | Automatically mid-task (for example, Plan → Agent on plan approval).

Select Mode settings, select mode for ex draw.io, then add, can be added to local workspace or made available for all workspaces(global). On adding mode , documentation of that mode will be made available for exact and optimum usage of that mode in mode settings of IBM Bob.

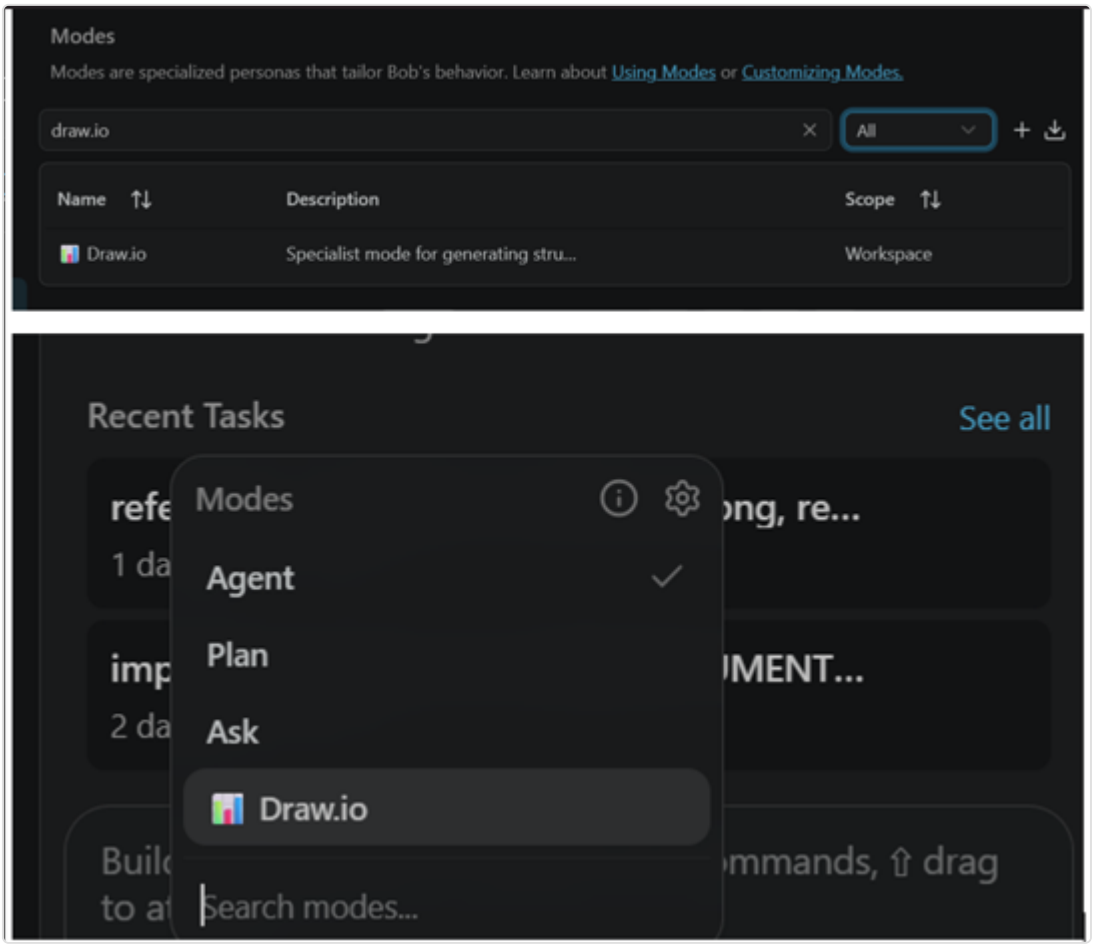


Figure: IBM Bob Custom Mode

IBM Bob permissions: Controlling what Bob can do

Understanding IBM Bob's operating modes is only part of the overall picture. Equally important is understanding its permission model, which determines not only what IBM Bob is technically capable of doing, but also what it is permitted to do and whether it must obtain user confirmation before taking an action.

The permission model is designed to provide controlled access to files, development tools, terminal commands, external systems, predefined skills, workflows, and task-management capabilities. Organizations can use the permission model to balance productivity with security by granting each mode only the capabilities required for its intended purpose.

IBM Bob's guidance is clear: Start with a restrictive configuration. Grant only the minimum permissions required and add further permissions when they become necessary. Auto-approval should never be enabled in a production environment.

Permissions in IBM Bob operate at two distinct levels.

Permission level	Purpose	Configuration impact
Mode-level permission	Determines which tool groups a particular mode is allowed to use.	Configured in custom_modes.yaml. If a tool group is not assigned to the mode, Bob cannot use the associated tools.
Auto-approve permission	Determines whether Bob can use an already permitted tool without requesting confirmation each time.	When auto-approval is disabled, Bob presents the proposed action and waits for user approval before proceeding.

This distinction is important. Granting a permission at the mode level makes a capability available to Bob, but it does not necessarily mean that Bob can exercise that capability automatically. Auto-approval provides the second layer of control.

For example, a mode might be permitted to execute terminal commands, while auto-approval for the execute group remains disabled. In this case, Bob can propose a command, but it must wait for the user to review and approve the command before running it.

Permission Groups at a Glance:

Permission group	Primary purpose	Risk level	General recommendation
read	Inspect files, directories, patterns, and code symbols	Medium	Foundational permission. Included by default in every mode.
edit	Create and modify files	High	Grant only when the mode must write code or update project artifacts.
execute	Run shell or PowerShell commands	High, highest-risk category	Use only in development environments. Keep auto-approval disabled.
mcp	Access configured Model Context Protocol servers	Medium-High	Grant for trusted external integrations. Auto-approve only fully reviewed servers.
skill	Load and activate SKILL.md instruction sets	Medium	Generally recommended because skills remain subject to existing tool permissions.
workflow	Launch predefined, structured workflows	Low-Medium	Useful for repeatable processes such as pull-request creation and code review.
todo	Maintain a visible task checklist	Low	Recommended for multi-step tasks and progress tracking.
subtask	Create visible, focused conversation threads	Low	Useful for organizing large or complex requests.
subagent	Spawn independent agents for focused work	Low	Useful when isolated exploration would otherwise consume the main context.
mode	Switch between operating modes	Low	Generally recommended to support smooth transitions during a task.

read: Read files and directories

Risk level: Medium

With the read permission, IBM Bob can understand the workspace and its codebase. Every standard mode, including Agent, Plan, and Ask, includes read access by default.

Capabilities enabled

When the read group is granted, Bob can inspect the workspace using the following tools:

Tool	Purpose
read_file	Displays file contents, optionally within specified line ranges.
glob	Locates files by name or path pattern, such as **/*.ts.
grep	Searches file contents using regular expressions.
list_files	Browses the directory structure.
GetSymbolsOverview	Provides an overview of code symbols.
FindSymbol	Locates a specific code symbol.
FindReferencingSymbols	Identifies symbols that reference another symbol.

When to grant it

The read permission should generally be granted in every mode. Without it, Bob cannot inspect the workspace or understand the codebase. It is therefore essential for almost every code-related activity.

Auto-approval considerations

Read access might still expose sensitive information. If a project contains credentials in configuration files, private keys, personally identifiable information, or other confidential data, organizations might choose to keep auto-approval disabled.

With manual approval enabled, the user can see which resource Bob intends to access before permitting the read operation.

The `.bobignore` file can also be used to prevent Bob from accessing specific files entirely.

Impact of not granting the permission

Without the read permission, Bob is effectively blind to the project. It cannot read a single file and can respond only to information entered directly into the chat. As a result, code analysis, code navigation, debugging, and most implementation-related tasks become impossible.

edit: Create and modify files

Risk level: High

With the edit permission, IBM Bob can create new files and modify existing project artifacts. This capability is central to Agent mode and to any workflow in which Bob is expected to implement code changes.

Capabilities enabled

Tool	Purpose
write_file	Creates a new file or completely rewrites an existing file.
apply_diff	Applies targeted changes to specific lines or sections.
insert_content	Inserts new content at a specified line.
search_and_replace	Finds and replaces text within a file.

Fine-grained control with fileRegex

The edit permission can be restricted by using a regular-expression pattern in `custom_modes.yaml`. With this pattern, an organization can control which file types Bob may modify.

For example: groups:

- read
- edit fileRegex: ".*.(js|ts)$" description: JavaScript and TypeScript files only

  In this configuration, Bob can edit only `.js` and `.ts` files. Any attempt to modify `.yaml`, `.env`, or another file type is blocked.

The same approach can be used to restrict Bob to SAP development artifacts such as `.abap` or `.cds` files.

When to grant it

Grant the edit permission when Bob must perform tasks such as:

- Implementing features
- Fixing bugs
- Refactoring code
- Creating new project artifacts
- Updating existing files
- Performing any task that requires Bob to write code

Agent mode requires this permission to fulfill its primary implementation purpose.

When not to enable auto-approval

Auto-approval should remain disabled in the following situations:

- Bob is working near production configuration files.
- The workspace contains secrets or sensitive information.
- Bob is being used for a code review where changes are not expected.
- Every proposed difference must be reviewed before it is saved.

With manual approval, Bob presents the proposed change and waits for the user to review it before the modification is applied.

Impact of not granting the permission

Without edit, Bob can analyze, plan, explain, and recommend changes, but it cannot modify any file. It can describe exactly what should be changed, after which the user must apply the changes manually.

This configuration is useful when creating a safe reviewer mode that is intended to assess code without introducing modifications.

execute: Run terminal commands

Risk level: High, representing the highest-risk permission category

With the execute permission, IBM Bob can run arbitrary shell or PowerShell commands. Because terminal access can affect the project, source-control repository, dependencies, operating system, and connected environments, this permission requires the strictest controls.

IBM's security guidance states: "Command line access is particularly dangerous, as it can potentially run harmful operations that could damage your system or compromise security."

Capabilities enabled

The execute group can be used to run commands such as:

Command category	Examples
Dependency installation	npm install, pip install
Build execution	mvn build
Source-control operations	git commit, git push
Test execution	pytest, npm test, jest
System operations	Any other permitted system-level shell or PowerShell operation

When to grant it

The execute permission might be granted in the following situations:

- Bob must build an application.
- Bob must run automated tests.
- Dependencies must be installed.
- A workflow requires command-line tools.
- A multi-step development process includes CLI operations.

This permission should be used only in development environments and should never be granted for production execution.

When not to enable auto-approval

`execute` is the most dangerous permission. Even when the mode has been granted access to the group, auto-approval should remain disabled. With auto-approval disabled, Bob displays each command and waits for confirmation before executing it. The user can review and, where supported, edit the proposed command before approving it.

Impact of not granting the permission

Without execute, Bob cannot run terminal commands. It can still generate command suggestions in text form, but the user must copy and execute those commands manually.

Plan mode and Ask mode intentionally exclude this capability because they are designed primarily for analysis, planning, and explanation rather than direct execution.

mcp: Access Model Context Protocol servers

Risk level: Medium-High

With the mcp permission, IBM Bob can call any configured Model Context Protocol server. MCP integrations extend Bob beyond the local workspace by connecting it to external systems, enterprise data sources, databases, and registered APIs.

Capabilities enabled

Examples of MCP-based integrations include:

MCP integration	Example use
SAP OData services	Query data from SAP S/4HANA.
Database connectors	Retrieve or process information from connected databases.
External APIs	Invoke APIs that have been registered as MCP tools.
Enterprise systems	Access connected systems involved in business or development workflows.

When to grant it

Grant the mcp permission in the following situations:

- A workflow requires external system integration.
- Bob must query SAP S/4HANA through SAP OData services.
- Bob must retrieve Jira tickets.
- Bob must invoke a registered custom API.
- Bob is being used with enterprise data sources.

When not to enable auto-approval

MCP servers might have broad access to external systems and might be capable of reading or writing enterprise data. When auto-approval is enabled, Bob can call those systems without requesting explicit confirmation for every invocation.

Auto-approval should therefore be enabled only for MCP servers that have been fully reviewed, trusted, and approved for the intended use.

Impact of not granting the permission

Without the mcp permission, Bob cannot call external tool servers. All MCP integrations are disabled.

If a task depends on live data from an external system, Bob can work only with information that the user manually provides in the conversation.

skill: Load and activate skills

Risk level: Medium

With the skill permission, IBM Bob can invoke `use_skill`, load a `SKILL.md` instruction set, and follow the specialized workflow defined by that skill.

Skills provide a mechanism for teaching Bob repeatable and domain-specific ways of working.

Capabilities enabled

Examples include:

Skill	Purpose
create-plan	Supports structured planning and is automatically loaded in Plan mode.
build-mcp-server	Guides the creation of an MCP server.
create-mode	Supports the creation of custom operating modes.
office-insights	Provides a specialized workflow for office-related insights.
Custom skills	Enables organization-specific workflows authored for Bob.

When to grant it

The skill permission can be granted in most modes. Skills offer high utility because Bob can follow consistent, reusable procedures.

Their risk is comparatively limited because a skill operates within the mode's existing tool constraints. A skill cannot grant Bob a capability that the current mode does not already possess.

When not to enable auto-approval

If users need to review the instructions contained in a skill before Bob follows them, auto-approval should remain disabled. Bob will then display the skill activation and wait for the user's approval.

Impact of not granting the permission

Without the skill permission, Bob cannot load any skills.

Plan mode's mandatory create-plan skill would fail to activate, which would prevent its core planning workflow from operating as intended. Any custom skills created by the organization would also be inaccessible.

workflow: Launch predefined workflows

Risk level: Low-Medium

With the workflow permission, IBM Bob can call `start_workflow` and trigger a named, curated, multi-step process.

Capabilities enabled

Examples of predefined workflows include:

- Creating a pull request from a Git difference
- Performing a code review against a base branch
- Running other structured and recurring tasks

When to grant it

Grant this permission when Bob is used for repeatable development processes, such as pull-request generation or automated code-review pipelines.

Impact of not granting the permission

Without workflow, Bob cannot trigger built-in workflows. It might still perform equivalent activities manually, one step at a time, but it cannot use the structured workflow runner.

todo: Update task todo lists

Risk level: Low

With the todo permission, IBM Bob can call `update_todo_list` and maintain a live Markdown checklist in the user interface.

The checklist can represent task status using the following states:

Status	Markdown representation
Pending	[ ]
In progress	[-]
Completed	[x]

When to grant it

This permission is recommended for any multi-step task. The todo list gives the user visibility into the following information:

- What Bob has already completed
- What Bob is currently doing
- What remains to be done

Impact of not granting the permission

Without todo, Bob cannot display live task progress.

This limitation might not matter for a simple, single-step request. However, for a complex implementation, users lose the integrated progress tracker and must track the task state themselves.

subtask: Create subtasks

Risk level: Low

With the subtask permission, IBM Bob can call `start_subtask` and create a separate, visible conversation thread for a focused area of work.

Each subtask can have the following components:

- Breadcrumb
- Conversation history
- Todo list

Users can follow the subtask and interact with it independently.

When to grant it

Grant the subtask permission for large or complex requests that benefit from dedicated, trackable workstreams.

For example, a broad request such as "implement this entire feature" might be divided into a design subtask and an implementation subtask.

Impact of not granting the permission

Without subtask, Bob must perform all work in the current conversation thread. Large tasks might become harder to follow, and the conversation can grow significantly.

No fundamental capability is lost. The primary impact is on organization, separation of work, and visibility.

subagent: Spawn subagents

Risk level: Low

With the subagent permission, IBM Bob can create an independent agent that runs in its own isolated context window. The subagent performs focused work and returns a summary to the main conversation.

Capabilities enabled

Subagent type	Access profile	Intended use
explore	Read-only and uses a lighter model	Codebase research and focused exploration
general	Full tool access and uses the default model	General-purpose delegated work

When to grant it

This permission is useful when Bob is working on a sufficiently complex task and exploring part of a large codebase would introduce too much irrelevant detail into the main context.

Bob uses subagents sparingly and only when the following three conditions are met:

- The delegated activity is self-contained.
- Performing the work in the main conversation would create significant context pollution.
- The activity cannot reasonably be completed in one or two tool calls.

Impact of not granting the permission

Without subagent, Bob must perform all exploration directly in the main conversation.

For large codebases, this can fill the available context with intermediate research and might reduce response quality during long-running tasks.

mode: Switch modes

Risk level: Low

With the mode permission, IBM Bob can call `switch_mode` and transition autonomously between operating modes during a task.

For example, Bob might begin in Plan mode, develop a solution plan, and then move to Agent mode after the plan is approved and ready for implementation.

When to grant it

This permission is generally recommended. Bob can adapt its operating mode as the task evolves without requiring the user to perform every transition manually.

Impact of not granting the permission

Without mode, Bob remains locked in the mode in which the task began. If the work later requires a different capability set, Bob must ask the user to switch modes manually.

Permission-Selection Guidance The appropriate permission configuration depends on the intended purpose of the mode.

Intended mode or activity	Recommended permissions	Permissions requiring stronger control
Codebase analysis	read, skill, todo	Consider manual approval for reads involving sensitive files.
Safe code reviewer	read, skill, todo, mode	Do not grant edit or execute.
Planning	read, skill, todo, mode	edit and execute are normally unnecessary.
Code implementation	read, edit, skill, todo, mode	Keep edit under review where sensitive files are present.
Build and test automation	read, edit, execute, skill, todo, mode	Keep execute auto-approval disabled.
SAP or external-system integration	read, mcp, skill, todo, mode	Grant mcp only for reviewed and trusted servers.
Large, multi-workstream implementation	read, edit, skill, workflow, todo, subtask, subagent, mode	Add execute or mcp only when required.
Production-related activity	Minimum permissions required for observation	Never enable auto-approval. Do not grant production execution capability.

Best practice: Start restrictive

IBM Bob's recommended approach is to start with the smallest possible permission set and extend it only when the purpose of the mode requires additional capabilities.

"Start restrictive. Begin with minimal permissions and add more only as needed. Never auto-approve in production."

A safe starting configuration for a custom mode is:

Groups:

- read
- skill
- mode

This baseline provides the following capabilities:

Permission	Reason for inclusion
read	Provides foundational workspace and codebase visibility.
skill	Offers low-risk, high-utility access to specialized workflows.
mode	Supports smooth transitions between operating modes.

Additional permissions should be introduced selectively:

Additional permission	Add only when
edit	The mode must create or modify files.
execute	The mode must build, test, install dependencies, or use command-line tools in a development environment.
mcp	The mode must access reviewed and trusted external systems or enterprise data sources.
workflow	The mode must launch a structured, predefined process.
todo	Multi-step task progress should be visible in the interface.
subtask	Complex work should be divided into separate visible threads.
subagent	Focused exploration should be isolated from the main context.

Key security principles

The permission model can be applied effectively by following these principles:

Apply least privilege. Grant only the tool groups required for the mode's intended purpose. Separate capability from approval. A mode may be permitted to use a tool while still requiring manual approval for every invocation. Treat terminal access as the highest-risk capability. Keep execute auto-approval disabled and limit its use to development environments. Review MCP servers before trusting them. MCP servers might read or write data in external systems. Use `fileRegex` to constrain editing. Restrict Bob to the file types relevant to the task. Protect sensitive files. Use `.bobignore` to prevent access to credentials, private keys, personally identifiable information, or other protected content. Use non-modifying modes for reviews. A reviewer mode can include read while excluding edit and execute. Use low-risk orchestration capabilities for complex work. Permissions such as todo, subtask, subagent, workflow, and mode improve structure and visibility without automatically granting file modification, command execution, or external-system access. Never auto-approve in production. Production actions should remain under explicit human control.

Key takeaways

IBM Bob’s permission model provides a structured way to control both capability and autonomy. Mode-level permissions determine which tool groups Bob can access, while auto-approval settings determine whether Bob must request confirmation before using those tools.

The safest configuration begins with read, skill, and mode, followed by the selective addition of higher-risk permissions only when they are essential. Permissions such as edit, execute, and mcp should receive particular scrutiny because with them Bob can modify files, run commands, or interact with external systems.

By combining least-privilege access, manual approval, file-type restrictions, trusted MCP configurations, and environment-specific controls, organizations can use IBM Bob productively while maintaining clear human oversight.

IBM Bob rules and skills: The synergy of rules and skills in enterprise code generation

As enterprise software engineering evolves under the paradigm of Generative AI, organizations managing SAP landscapes face a dual challenge: accelerating development velocity while strictly adhering to the SAP Clean Core strategy. Modern SAP ecosystems—centered on ABAP Cloud, the RESTful Application Programming Model (RAP), and Core Data Services (CDS)—leave no room for legacy patterns, unmanaged custom code, or non-released standard APIs.

Generic code-generation tools often fail in such constrained environments, producing syntactically acceptable but architecturally non-compliant code. IBM Bob resolves this friction through a dual-engine architecture composed of Rules (persistent governance) and Skills (contextual execution playbooks).

This section details the architectural significance, mechanics, and practical synergy of Rules and Skills within SAP enterprise development.

Conceptual architecture

The Dual-Engine Model To establish governance while maximizing developer productivity, IBM Bob decouples ambient compliance from task-specific workflows.

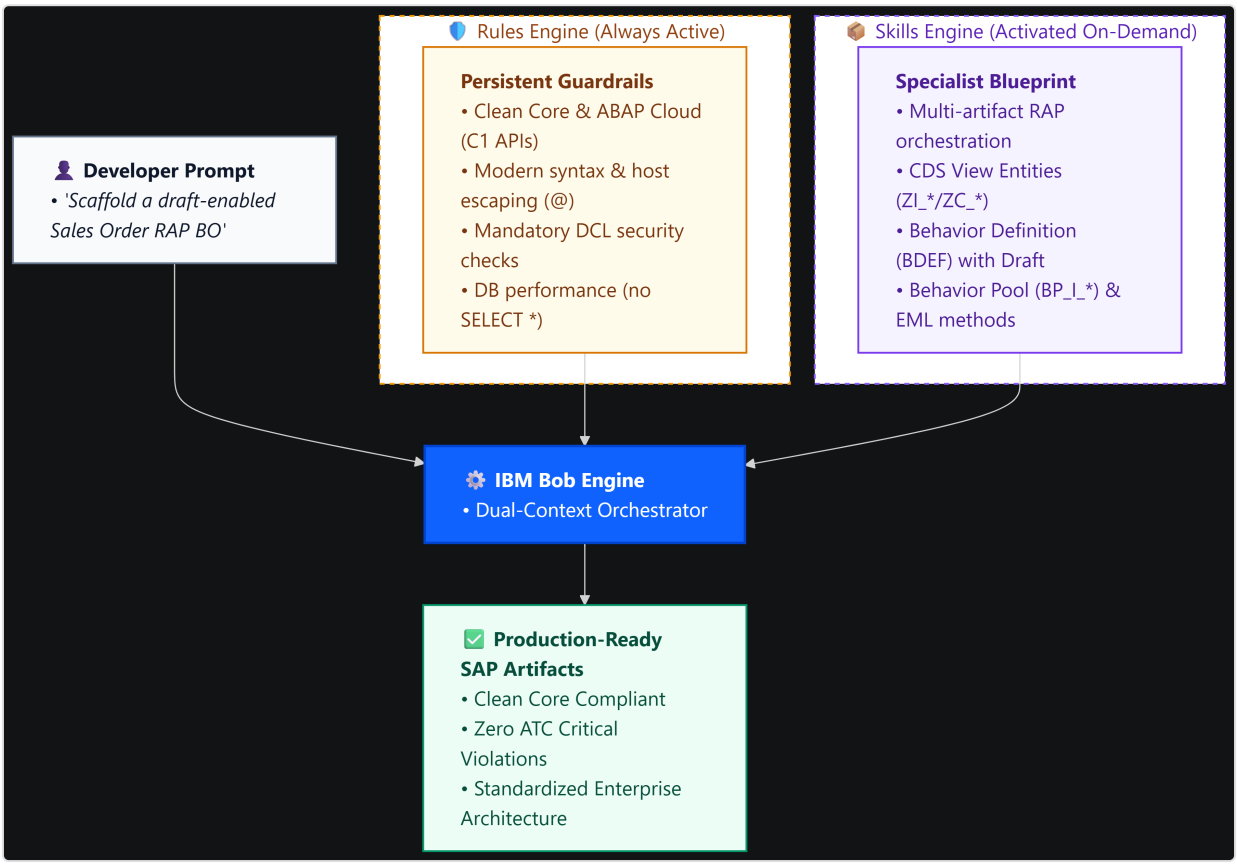


Figure: CleanCore by Design: Bridging Rules and Skills in IBM Bob

Persona perspective

The Enterprise Journey To understand the practical impact, consider two key enterprise roles working within an S/4HANA modernization program:

The Enterprise SAP Architect (Governance and Quality Assurance): Responsible for ensuring zero-regression migrations, setting corporate naming standards, and enforcing ABAP Test Cockpit (ATC) compliance across distributed development teams. The SAP Application Developer (Feature Delivery and Implementation): Responsible for translating business requirements into end-to-end transactional applications, data services, and extensions.

Rules: The Ambient Guardrails of the Enterprise Architect Architectural Definition In IBM Bob, Rules are persistent configuration sets (defined under `.bob/rules/`) that remain continuously active across every prompt and code generation cycle. They establish non-negotiable boundaries, coding standards, and security mandates.

Significance in SAP Code Generation From the Enterprise Architect's perspective, Rules guarantee that AI-assisted output never drifts from organizational compliance policies:

Clean Core and ABAP Cloud Enforcement: Restricts code generation strictly to ABAP Cloud language scope. Prohibits direct access to standard base tables (e.g., `MARA`, `VBAK`, `KNA1`), enforcing consumption through released CDS Interface Views (e.g., `I_Product`, `I_SalesOrder`, `I_Customer`). Eliminates deprecated statements (such as `MOVE`, `COMPUTE`, `CHECK` outside loops, header lines, and obsolete function modules). HANA Database Optimization and Code Pushdown: Prevents database anti-patterns such as `SELECT *`, nested `SELECT...ENDSELECT` loops, or unbounded memory operations. Mandates strict SQL syntax mode (`STRICT_2020` or higher) and required host variable escaping with `@`. Enterprise Security and Authorization: Automatically incorporates mandatory Data Control Language (DCL) annotations for CDS views and AUTHORITY-CHECK statements in custom implementations. Representative Rule Configuration (`.bob/rules/sap-clean-core.md`)

Enterprise SAP Clean Core and ABAP Cloud standards

- Target Scope: ABAP Cloud / S/4HANA Cloud and 2022+ On-Premise
- Syntax and Integrity:
  - Use modern syntax: inline declarations DATA(...), table expressions, and string templates.
  - Reject all obsolete keywords (MOVE, TABLES, HEADER LINES).
  - Use OpenSQL/ABAP SQL with strict mode and host variable escaping (@).
- Clean Core Governance:
  - Do not reference non-released standard SAP database tables directly.
  - Utilize SAP released CDS Interface Views (I_*) and released APIs (C1 contract).
- Security:
  - Ensure every CDS View Entity has an associated DCL or explicit `@AccessControl.authorizationCheck: #NOT_REQUIRED` with documented justification.

Skills: The Domain Playbooks of the Application Developer Architectural Definition Skills are modular, on-demand instruction packages (`SKILL.md` along with schema templates and design patterns) stored under `.bob/skills/`. Unlike Rules, Skills are loaded dynamically when a user's prompt matches a complex, multi-step engineering scenario.

Significance in SAP Code Generation Modern SAP development rarely involves isolated scripts; it requires the structured generation of interdependent layers. The Application Developer relies on Skills to scaffold entire architectural patterns accurately:

End-to-End Generation Sequence

The SAP RAP Skill orchestrates artifact generation through a dependency-driven sequence, ensuring that each artifact is aligned with the underlying data model, business behavior, service contract, and user interface. IBM Bob follows the sequence below to generate a cohesive RAP business object while minimizing inconsistencies across dependent artifacts.

Generation workflow

1. **Requirement Analysis and Business Object Definition:** Analyze the functional requirements, identify the business object structure, determine root and child entities, and establish keys, relationships, transactional requirements, and draft-handling needs.
2. **Database Persistence and Draft Tables:** Generate the required database tables and draft persistence structures, where applicable, based on the business object design.
3. **Interface CDS View Entities (ZI_*):** Create root and child interface views, define associations and compositions, and establish consistent keys and semantic relationships.
4. **Projection and Consumption CDS Views (ZC_*):** Generate projection views to expose the required business object elements and define the consumption layer for application-specific use.

5. **Behavior Definition (BDEF):** Define supported operations, transactional behavior, locking, authorization, validations, determinations, and draft handling, as required by the use case.
6. **Behavior Implementation Classes (BP_I_*):** Generate the corresponding implementation classes, including local handler and saver classes, to implement business rules and transactional logic.
7. **EML Implementation and Error Handling:** Generate and validate READ ENTITIES and MODIFY ENTITIES operations, including appropriate handling of FAILED and REPORTED responses and their relevant components.
8. **Service Definition (SRVD):** Define the business entities and operations to be exposed through the application service.
9. **Service Binding (SRVB):** Configure the service binding for the required protocol and deployment scenario, such as OData V4 where applicable.
10. **Metadata Extensions and UI Annotations:** Generate metadata extensions and annotations, including @UI.lineItem and @UI.identification, to support SAP Fiori Elements list reports, object pages, and related user interactions.
11. **Cross-Artifact Consistency Validation:** Verify that entity names, keys, associations, compositions, behavior definitions, implementation references, and service exposure remain consistent across all generated artifacts.
12. **Activation, Testing, and Refinement:** Activate the generated repository objects, resolve syntax and dependency errors, execute relevant functional and transactional tests, and refine the artifacts based on validation results.

Expected outcome: A coordinated set of SAP RAP artifacts generated in dependency order, with consistent data modeling, transactional behavior, service exposure, and Fiori Elements integration. IBM Bob uses the SAP RAP Skill to guide artifact generation and validation; successful activation and runtime correctness remain subject to the target ABAP environment, release-specific constraints, and applicable development and testing procedures.

Comparison and Operational Alignment The following matrix summarizes the distinct functions and collaborative relationship between Rules and Skills:

Dimension	Rules (The Guardrails)	Skills (The Playbooks)
Operational Lifecycle	Ambient / Continuously active across all workspace sessions	On-demand / Triggered by specific semantic intents
Primary Objective	Compliance, governance, Clean Core alignment, security	Multi-tier orchestration, scaffolding, design pattern execution
Architectural Scope	Horizontal across every line of generated code	Vertical into specialized domains (RAP, CDS, AMDP, CAP)
Target Persona	Enterprise Architect and Quality Lead	Application Developer and Technical Lead
Enterprise Value	Prevents technical debt and ATC compliance failures	Accelerates end-to-end delivery of complex business models

Key takeaways

In enterprise SAP development, generative capability without strict governance increases technical debt and migration risk. By separating Rules (continuous architectural guardrails) from Skills (modular, task-specific recipes), IBM Bob ensures that developers achieve unprecedented velocity while maintaining full adherence to SAP Clean Core, ABAP Cloud, and enterprise security standards.

Enhancing SAP ABAP code with intelligent assistance (IBM Bob skill)

A Bob skill is a defined capability, such as performing ATC checks, generating ABAP code, or orchestrating workflows, that can be invoked to solve a specific task.

How IBM Bob skills enhance ABAP code quality

IBM Bob skills can help teams make ABAP development practices more consistent, reviewable, and easier to apply across a project. In this context, a skill is a reusable package of guidance that describes how to perform a particular type of work. A skill may provide instructions, preferred patterns, examples, checks, and expected outputs for a defined activity for example, reviewing an ABAP class, preparing a transport request, or creating unit tests.

For teams that use AI-assisted development, skills are particularly useful because they provide task-specific context. Rather than relying on generic programming advice, a skill can guide work according to the conventions and technical expectations of a particular SAP landscape. The result is not an automatic guarantee of quality; ABAP developers and reviewers remain accountable for validating the code. However, well-designed skills can make quality practices more repeatable and can reduce the likelihood that important checks are overlooked.

Understanding skills for beginners

A skill is best understood as an operational playbook. It answers questions such as:

- What is the purpose of this development task?
- What information should be examined before making a change?
- Which practices are preferred?
- Which risks should be checked?
- What should the finished result include?

For example, an “ABAP code review” skill might instruct a developer or assistant to inspect naming consistency, public method contracts, database access, authorization checks, exception handling, test coverage, and documentation. A “transport readiness” skill might focus on activation status, syntax and extended-program checks, object dependencies, test evidence, and the completeness of the transport request.

Skills are most effective when they are focused. A broad instruction such as “write good ABAP” is difficult to apply consistently. A focused skill such as “refactor database access in an ABAP class” can specify the relevant decisions: whether data is selected in a loop, whether the selected fields are minimal, whether result sets are bounded, whether joins are appropriate, and whether the refactoring preserves existing behavior.

Skills and rules serve different purposes

Skills and Rules are complementary, but they are not the same thing.

Rules define mandatory or persistent constraints. They express requirements that should be followed consistently across relevant work. Examples include:

- Use project-approved naming conventions.
- Do not bypass authorization checks for business actions.
- Do not include secrets, passwords, or production credentials in source code.
- Require peer review before transporting changes to a quality system.
- Use approved exception classes for application errors.

Rules answer the question: “What must or must not be done?”

Skills describe how to carry out a task effectively within those constraints. They may include a sequence of steps, examples, review prompts, and suggested deliverables. For instance, a performance-review skill can explain how to identify SELECT statements inside loops, assess table buffering, examine use of indexes, and validate the result with realistic test data.

Skills answer the question: “How should this work be performed?”

This distinction is important for governance. Rules establish the guardrails; skills help developers navigate within them. A team may have a mandatory rule requiring ABAP Unit tests for new business logic, while a testing skill explains how to structure testable classes, create test doubles where appropriate, and organize test methods using arrange-act-assert logic.

Improving naming and code readability

Clear naming is a foundational ABAP quality practice. Names should communicate the purpose of a variable, method, class, interface, or database operation without requiring a reader to reverse-engineer the implementation.

A naming skill can guide developers to use names that reflect business meaning and technical role. For example, lv_sales_order_id is more informative than lv_id, and get_open_items is clearer than get_data. The skill can also remind developers to distinguish values, collections, flags, and object references consistently, according to the team’s convention.

Naming guidance should not become a mechanical prefix exercise. The objective is readable code. A method named validate_delivery_block conveys both intent and scope. A local variable named lt_unreleased_orders immediately tells the reader that it is an internal table containing orders that have not been released.

Consistent naming improves maintainability because it reduces cognitive load during troubleshooting, code review, and future enhancement work. It also makes automated review prompts more useful: a reviewer can quickly identify methods whose names do not match their behavior or whose parameters are too vague to support safe reuse.

Encouraging modularity and clear responsibilities

ABAP code quality improves when units of code have clear, limited responsibilities. Skills can promote modular design by encouraging developers to separate orchestration, validation, persistence, and external integration concerns.

For object-oriented ABAP, this often means designing classes with focused public methods and moving complex logic into private helper methods or collaborating objects. A method that validates an order, reads customer data, calculates a discount, updates database tables, sends an outbound message, and formats a report is difficult to test and difficult to change safely. A modularity skill can prompt the developer to divide these responsibilities into smaller units with explicit contracts.

For procedural ABAP, the same principle applies through well-named FORM routines, function modules, or includes, while recognizing that newer development should generally favor appropriate object-oriented structures. The goal is not to create excessive layers or tiny methods with no meaningful purpose. The goal is to create units that can be understood, tested, and changed independently.

A useful skill prompt is: Can the method’s responsibility be described in one sentence without using “and then”? If not, the method may need to be decomposed.

Strengthening error handling

Business applications must handle expected failures in a predictable way. In ABAP, quality issues frequently arise when code ignores return values, relies only on sy-subrc without meaningful handling, suppresses exceptions, or exposes technical failures directly to users.

An error-handling skill can establish a consistent approach for the following tasks:

- Checking the outcome of database operations and function calls.
- Raising meaningful, typed exceptions for application-level failures.
- Preserving technical details for diagnostics while providing appropriate messages to business users.
- Avoiding empty CATCH blocks or generic error suppression.
- Defining whether a failure should stop processing, skip an item, or be collected for later reporting.

For class-based development, exception classes make method contracts clearer. A method that can reject an invalid input should state that behavior through an appropriate exception or return structure. Callers can then handle the failure deliberately rather than discovering it indirectly through inconsistent state or an unexpected dump.

Skills can also encourage developers to distinguish between validation errors, authorization failures, integration failures, and technical infrastructure failures. These categories often require different messages, logging, retry behavior, and operational responses.

Integrating authorization checks into design

Authorization should be treated as part of business logic rather than as a final check added shortly before transport. A skill focused on authorization can guide developers to identify the protected action, relevant authorization object, organizational scope, and expected behavior when authorization is missing.

For example, a custom transaction that changes pricing conditions should verify that the user is authorized to perform that action for the relevant sales organization or other applicable organizational level. A check performed only at the user interface may not be sufficient if the same business function can be called through another entry point, such as a report, service, background job, or remote-enabled function module.

The skill should encourage developers to observe the following practices:

- Identify where the business action is actually executed.
- Apply the authorization check at an appropriate enforcement point.
- Use project-approved authorization objects and values.
- Return a clear and safe message when access is denied.
- Test both authorized and unauthorized scenarios.

Authorization design also benefits from documentation. A brief note explaining why a check is required and which business action it protects helps functional, security, and audit stakeholders review the implementation.

Improving database performance

Database access is one of the most important areas of ABAP quality, especially in high-volume business processes. A performance skill can guide developers toward efficient Open SQL patterns without treating performance as a collection of simplistic prohibitions.

The following review points are commonly addressed:

- Avoiding database SELECT statements inside loops when a set-based query can be used.
- Selecting only the fields needed by the application.
- Restricting result sets with appropriate conditions.
- Considering joins or other set-oriented operations where they improve clarity and reduce round trips.
- Avoiding unnecessary repeated reads of the same data.
- Evaluating internal-table access patterns for large data volumes.
- Considering the effect of table buffering only when it is relevant and understood.
- Validating assumptions with realistic volumes and available performance-analysis tools.

A skill should not insist that every query be rewritten as a complex join. Readability and correctness remain important. The appropriate design depends on the data model, expected volume, transactional context, indexing, and functional requirements. The important contribution of the skill is to make performance considerations explicit early in development and during review.

Building testing into ABAP development

Testing skills help move quality assurance from a late-stage activity into the development process. For ABAP classes containing business logic, ABAP Unit tests can verify expected behavior quickly and repeatedly. Skills can guide developers to design code that is testable: dependencies should be isolated where practical, business logic should not be tightly coupled to screen processing, and database or external-service interactions should be separated from decision logic when feasible.

A practical testing skill can ask developers to cover the following scenarios:

- Normal business scenarios
- Boundary conditions
- Invalid inputs
- Authorization-denied cases
- Expected technical or integration failures
- Regression scenarios related to a defect fix

Tests should be meaningful rather than merely increasing a coverage figure. A test that verifies an exception is raised for an invalid sales order status provides more value than a test that only executes a method without checking its outcome.

For legacy code that is difficult to test directly, a skill can recommend incremental improvement. Developers might first extract a small calculation or validation routine into a testable class, add tests for that routine, and then expand coverage as the code evolves. This approach is often more practical than attempting a large-scale rewrite solely to introduce tests.

Making documentation useful

Documentation is often most valuable when it records decisions that are not obvious from the code itself. A documentation skill can encourage developers to explain business rules, assumptions, integration dependencies, authorization decisions, unusual performance considerations, and operational behavior.

Good documentation does not repeat every line of implementation. Instead, it answers questions that a future maintainer or reviewer would otherwise need to investigate. For example:

- Why is a particular status excluded from processing?
- Why does a database query use a specific selection condition?
- Which interface receives the outbound message, and what happens if it fails?
- Which authorization object protects the update action?
- Which customizing or configuration prerequisite is required?

At the code level, public classes and methods benefit from concise descriptions of purpose, inputs, outputs, side effects, and exceptions. At the transport level, a clear change description helps reviewers understand what was altered and how it was validated.

Supporting transport readiness

Transport readiness is more than assigning changed objects to a transport request. A transport-readiness skill can provide a repeatable checklist before changes leave a development environment.

A typical checklist includes the following items:

- All changed repository objects are activated.
- Syntax checks and applicable extended program checks have been completed.
- Related objects and dependencies are included in the appropriate transport requests.

- Required customizing changes are identified and handled according to the project's transport process.
- Relevant unit, integration, or regression tests have been executed.
- Authorization, error handling, and logging behavior have been reviewed.
- Transport descriptions clearly identify the business purpose and scope of the change.
- Any deployment instructions, prerequisites, or rollback considerations are documented.

The precise checklist should be adapted to the organization's SAP change-management process. For example, some landscapes require formal approvals, segregation-of-duties controls, or release documentation outside the ABAP development environment. A skill does not replace those controls; it helps ensure that developers prepare their changes in a way that supports them.

Practical example: Improving an order validation routine

Consider an ABAP routine that processes sales orders and checks whether each order is eligible for release. The original implementation contains a `SELECT SINGLE` inside a loop, uses generic variable names, returns only `sy-subrc`, and has no explicit authorization check.

```
LOOP AT lt_orders INTO DATA(ls_order).

SELECT SINGLE status

FROM zsales_order

WHERE order_id = @ls_order-id

INTO @DATA(lv_status).

IF sy-subrc = 0 AND lv_status = 'OPEN'.

" Release order

ENDIF.

ENDLOOP.
```

A skill for ABAP code quality would prompt the developer to address several concerns:

- **Naming:** Replace generic names with names that express their purpose.
- **Performance:** Avoid one database round trip per order where a set-based read is possible.
- **Modularity:** Separate data retrieval, authorization, eligibility validation, and release processing.
- **Error handling:** Define how missing or inconsistent order data is handled.
- **Authorization:** Verify that the user is permitted to release orders.
- **Testing:** Add tests for eligible, ineligible, missing, and unauthorized cases.
- **Documentation:** Record the business definition of "eligible for release."

A refactored design might use a service class with a public method such as `release_eligible_orders`. The method can first verify authorization, retrieve all relevant order statuses in one query, validate each order through a dedicated method, and return a structured result describing released orders, skipped orders, and failures.

The following simplified example illustrates the direction of the refactoring:

```
METHOD release_eligible_orders.

AUTHORITY-CHECK OBJECT 'Z_SO_REL'

ID 'VKORG' FIELD iv_sales_organization

ID 'ACTVT' FIELD '02'.

IF sy-subrc <> 0.

RAISE EXCEPTION TYPE zcx_order_release

  EXPORTING

    textid = zcx_order_release=>not_authorized.

ENDIF.

SELECT order_id, status
FROM zsales_order
FOR ALL ENTRIES IN @it_order_ids
WHERE order_id = @it_order_ids-table_line
INTO TABLE @DATA(lt_order_statuses).

LOOP AT it_order_ids INTO DATA(lv_order_id).
  READ TABLE lt_order_statuses
  WITH KEY order_id = lv_order_id
  INTO DATA(ls_order_status).

IF sy-subrc <> 0.

  APPEND VALUE #( order_id = lv_order_id

                  result   = 'NOT_FOUND' ) TO rt_results.
  CONTINUE.

ENDIF.

IF ls_order_status-status <> 'OPEN'.

  APPEND VALUE #( order_id = lv_order_id

                  result   = 'NOT_ELIGIBLE' ) TO rt_results.
  CONTINUE.

ENDIF.

release_order( lv_order_id ).

APPEND VALUE #( order_id = lv_order_id

                result   = 'RELEASED' ) TO rt_results.
ENDLOOP.

ENDMETHOD.
```

This example remains simplified. In production code, the developer should consider empty input handling before `FOR ALL ENTRIES`, transaction boundaries, locking, update behavior, exception propagation, and the data model's actual authorization requirements. Nevertheless, it demonstrates how a skill transforms a vague quality objective into concrete review and implementation actions.

Applying skills as a team practice

The value of IBM Bob skills is greatest when teams treat them as living engineering assets. A skill should be reviewed and updated as coding standards, SAP releases, project architecture, and operational lessons evolve. Teams can begin with a small set of focused skills—such as ABAP code review, ABAP Unit testing, performance review, and transport readiness—and refine them using real development feedback.

Skills should also be used consistently across the delivery lifecycle. Developers can apply them while implementing a change, reviewers can use them during peer review, and release coordinators can use them as part of transport preparation. This shared approach makes expectations more visible and reduces variation caused by individual habits.

Ultimately, skills support ABAP quality by making sound development practices easier to apply repeatedly. They do not replace experienced developers, architectural judgment, security review, testing, or formal governance. Instead, they provide a practical structure for bringing those disciplines into everyday ABAP work.

Governance, constitution, and AI guardrails for SAP development

A Bob rule enforces governance logic, ensuring compliance with clean core principles, coding standards, and enterprise policies during execution.

As organizations adopt IBM Bob for AI-assisted SAP development, a critical governance question emerges: how does the organization ensure that AI-generated code consistently follows the same naming conventions, security practices, and quality standards that apply to code written by hand? Without an enforced set of standards, AI-assisted development can introduce inconsistency at scale — the same risk that ungoverned human development creates, but potentially faster and across a wider surface area.

IBM Bob addresses this through **Bob Rules**: a configurable governance layer that functions as a project constitution. Rules are defined once, stored under version control alongside the codebase, and applied automatically and consistently every time Bob generates or modifies code — regardless of the prompt used or the developer issuing it.

This section defines the governance model for Bob Rules in SAP environments. It explains what belongs in a Bob Rule and what does not; documents the naming, coding, documentation, error-handling, testing, and transport standards that a typical SAP organization should enforce; and provides a step-by-step implementation guide for structuring rules by technology and development mode (ABAP, SAP UI5, and Node.js/CAP). For senior management, this section establishes the control framework that makes AI-assisted development auditable and compliant with existing SAP development governance. For architects, it defines the recommended rule architecture. For developers, it provides ready-to-use rule content and file structures.

Bob rules as the project constitution

When implementing an enterprise-grade, AI-ready development environment using IBM Bob, the organization must design a clear governance, guardrail, and constitution layer before AI-assisted development begins at scale. Bob Rules are the mechanism through which this layer is built.

A Bob Rule is a **global system context** that remains active at all times. Rules are configured under the `.bob` folder as custom instructions and guidelines that define how IBM Bob behaves when working on a given project. They function as a project constitution that Bob follows consistently across every interaction, independent of the specific prompt a developer issues.

Bob Rules should contain only standards that are **common across the entire team** — for example, naming conventions, mandatory code headers, or security requirements. They should never contain content that is specific to an individual team or user, such as personal preferences or a narrowly scoped coding skill; that type of content belongs in a different configuration mechanism, not in the Bob Rule. Because rules are invoked automatically and unconditionally, anything placed in a Bob Rule is enforced on every prompt, for every user, in every session.

Bob rule examples by technology

SAP ABAP development

- Enforce SAP naming conventions (Z*/Y* prefixes, Hungarian notation)
- Mandate code standards (ABAP Clean Code, performance patterns)
- Control object types (prefer classes over function modules)
- Define transport management standards (development class rules, package structure)
- Enforce security practices (authority checks, SQL injection prevention)
- Standardize error handling (exception classes, message handling)

SAP UI5 development

- Component structure (MVC pattern enforcement)
- Naming conventions (controller names, view IDs, model names)
- OData service patterns (annotation usage, batch requests)
- Fiori guidelines (UX consistency, accessibility)
- Performance rules (lazy loading, data-binding optimization)

Node.js / CAP development

- Service definition standards (CDS naming, entity relationships)
- API design patterns (RESTful conventions, error responses)
- Authentication and authorization (JWT handling, role-based access)
- Database patterns (HANA-specific optimizations)

Defining Bob rules for SAP projects

A Bob Rule can be authored as a simple Markdown file or as an XML file. The examples below illustrate typical rule content for ABAP and UI5 development.

ABAP development rule — Example

- All custom objects *must* start with `Z` or `Y` prefix
- Use package hierarchy: `Z<PROJECT>_<MODULE>_<SUBMODULE>`
- Development class naming: `Z<PROJECT>_<PURPOSE>`
- Transport requests *must* include descriptive text

UI5 development rule — Example

- Follow namespace pattern: `com.company.project.module`
- Component structure: `Component.js`, `manifest.json`, i18n files
- View naming: `<EntityName>.view.xml`
- Controller naming: `<EntityName>.controller.js`

Different naming convention rule examples

ABAP objects

Table: Standard ABAP object naming conventions.

Object Type	Convention	Example
Programs	Z__	Z_SALES_ORDER_REPORT
Classes	ZCL__	ZCL_SALES_ORDER_PROCESSOR
Interfaces	ZIF__	ZIF_SALES_ORDER_HANDLER
Function Groups	Z__	Z_SALES_UTILITIES
Tables	Z_	ZSALESORDER_LOG
Data Elements	Z_	ZSALES_ORDER_ID

Variable naming (Hungarian notation)

Table: Standard ABAP variable and parameter naming conventions.

Category	Convention	Notes
Local variables	lv_<name>	for example, lv_order_number
Global variables	gv_<name>	
Parameters	iv_<name> / ev_<name> / cv_<name>	importing / exporting / changing
Tables	lt_ / gt_ / it_ / et_<name>	local / global / importing / exporting
Structures	ls_ / gs_ / is_ / es_<name>	local / global / importing / exporting
Objects	lo_ / go_ / io_ / eo_<name>	local / global / importing / exporting
Field symbols	<fs_name>	
Constants	lc_ / gc_<name>	local / global

UI5 naming

Table: Standard SAP UI5 naming conventions.

Category	Examples
Models	oDataModel, jsonModel, i18nModel
Controls	oTable, oButton, oInput
Views	CustomerList.view.xml, CustomerDetail.view.xml
Controllers	CustomerList.controller.js
Fragments	CustomerDialog.fragment.xml

Coding standards and best practices rule

ABAP Clean Code principles

- Never use `SELECT *` — always specify fields explicitly
- Always use modern ABAP syntax (inline declarations, constructor expressions)
- Prefer classes over function modules for new development
- Exception handling is mandatory — no empty `CATCH` blocks
- Always add meaningful comments for complex logic
- Avoid nested `IF` statements beyond three levels — refactor into methods
- Limit method length to a maximum of 50 lines
- Always run the pretty printer before activation

Performance rules

- Use `SELECT SINGLE` when retrieving a single record
- Use `FOR ALL ENTRIES` with proper existence checks (not `lt_table IS INITIAL`)
- Avoid `SELECT` statements inside loops — use internal tables instead
- Use database views instead of nested `SELECT` statements
- Implement buffering for frequently accessed tables
- Use HANA-optimized code (CDS views, AMDP) where available

Security requirements

- Always implement authority checks for sensitive operations
- Never concatenate SQL strings — use prepared statements
- Validate all user inputs
- Log all critical operations
- Encrypt sensitive data held in custom tables

Documentation requirements

ABAP documentation

Every class must include a class-level comment describing its purpose. Every public method must include the following elements:

- A purpose description
- Parameter descriptions
- A return value description
- Example usage, for complex methods
- Author and creation date

Code comments

- Use `" *` for single-line comments
- Use `" !` for ABAP Doc comments
- Comment *Why*, not *What* — code should be self-explanatory
- Update comments whenever the underlying code changes

Technical specifications

A technical specification document should be created for each major development effort, including: purpose, design, data model, interface details, and test cases.

Error handling and logging

Exception management

- Always use exception classes (the `CX_*` hierarchy)
- Create custom exception classes using the pattern `ZCX_<PROJECT>_<PURPOSE>`
- Never suppress exceptions silently
- Log exceptions together with relevant context information
- Provide meaningful, actionable error messages to end users

Testing requirements

Unit testing

- Unit tests are mandatory for all business logic classes
- Minimum required test coverage is 80%
- Use the ABAP Unit framework
- Name test classes using the pattern `ltc_<class_name>`
- Mock external dependencies

Integration testing

- Test end-to-end scenarios
- Include error scenarios in test coverage
- Document test cases and their results

Transport and deployment

Development process

- Always create a transport request with descriptive text
- Group related objects within the same transport request
- Include supporting documentation in the transport request text
- Test in the development system before releasing
- Require peer review before transport release

Deployment guidelines

- Follow the standard `DEV → QAS → PRD` promotion path
- Obtain formal approval before production deployment
- Schedule deployments within approved maintenance windows
- Prepare a rollback plan for every deployment
- Document all deployment steps

Implementation methods for Bob rules

Bob Rules can be implemented through three complementary mechanisms, described below. The right mechanism depends on the size of the rule set, how frequently it changes, and whether it applies globally or only within a specific development mode.

Method 1

Global Rule File (Markdown)

Create a Markdown rule file at:

```
.bob/rules/rule.md
```

Bob automatically injects every file found in `.bob/rules/` into every conversation, across all modes. All guardrails defined in this location are active on every prompt; no trigger phrase is required.

Method 2

Dedicated Rules File (XML, by Domain)

Rules can be compartmentalized into separate folders by technical domain. For example, all ABAP-related rules can be placed inside a `rules-code-abap` folder, which ensures that these rules are invoked whenever Bob encounters a prompt related to ABAP code generation. For example:

```
.bob/Rules and Guardrails/rules-code-abap/abap documentation rule.xml

.bob/Rules and Guardrails/rules-code-cap-backend/1_cap_model_cds.xml
```

Method 3

Mode-Specific Rules

Key convention: The directory name must exactly match the mode's `slug`. For a mode with `slug: abap-mode`, the corresponding directory is `.bob/rules-abap-mode/`. For a mode with `slug: ui5-mode`, the directory is `.bob/rules-ui5-mode/`.

When to use mode-specific rules

There are three clear scenarios in which mode-specific rules are the appropriate mechanism.

Rules already live in external files

Enterprise projects commonly maintain coding standards as XML, Markdown, or text documents, version-controlled alongside the codebase. In SAP ABAP projects, documentation standards, naming conventions, and performance rules are frequently maintained as XML policy files. Rather than copying this content into `customInstructions`, the source file can be referenced directly using Bob's `@file` import syntax.

Rules are large or frequently updated

Embedding large volumes of XML rules directly inside `custom_modes.yaml` makes the configuration file difficult to read and maintain. A `rules-{mode}/` directory keeps the YAML configuration clean and allows rules to evolve independently — for example, a UI5 project-structure rule can be updated without touching the YAML configuration at all.

Different teams own different rule files

An ABAP basis team might own performance and security rules, while a development team owns documentation standards. Multiple files placed in the same `rules-{mode}/` directory are loaded in alphabetical order, allowing each team to maintain its own file without merge conflicts.

Benefits at a glance

Table: Comparing inline `customInstructions` with a `rules-{mode}/` directory.

Approach	Maintainability	Separation of Concerns	File Reuse
Rules inside customInstructions	Becomes unwieldy for large rule sets	Rules and mode configuration are mixed together	Content must be duplicated per mode
rules-{mode}/ directory	Clean YAML; rules kept in dedicated files	Each concern is isolated in its own file	@file can point to a single shared source

Project layout example

A typical SAP full-stack project using both ABAP and UI5 modes is organized as follows:

```
.bob/

├── custom_modes.yaml
├── Rules and Guardrails/
│   ├── rules-code-abap/
│   │   ├── abap documentation rule.xml
│   │   └── abap_comprehensive_standards rule.xml
│   └── rules-code-ui5-frontend/
│       ├── 8_ui5_project_structure.xml
│       └── 9_ui5_all_frontend_rules.xml
├── rules-abap-mode/
│   ├── 01-abap-documentation-rule.md
│   └── 02-abap-comprehensive-standards.md
└── rules-ui5-mode/
├── 01-ui5-project-structure.md
└── 02-ui5-frontend-rules.md
```

> **Tip:** Files in a `rules-{mode}/` directory are loaded alphabetically. Prefix filenames with numbers (`01-`, `02-`) to control load order explicitly.

Setting up mode-specific rules — Step by step

Follow these steps to configure a dedicated `rules-{mode}/` directory and link your modular rule files to a custom mode in IBM Bob:

Step 1: Identify the mode slug

Open `.bob/custom_modes.yaml` and locate the `slug` of the mode to be targeted.

```
#### # .bob/custom_modes.yaml

customModes:

- slug: abap-mode   *# ← this is your slug*

name: ABAP Mode

...
```

Step 2: Create the rules directory

Create a directory in `.bob/` named `rules-{slug}/`:

```
mkdir .bob/rules-abap-mode
```

Step 3: Create a rule file that imports the XML source

Use Bob's `@file` import syntax. Bob interprets the `@` prefix as an instruction to inline the content of the referenced file into context.

```
# .bob/rules-abap-mode/01-abap-documentation-rule.md

@.bob/Rules and Guardrails/rules-code-abap/abap documentation rule.xml
```

That single line is the entire file. Bob resolves the path relative to the workspace root and injects the referenced XML content when the mode is activated.

Step 4: Add additional rule files as needed

To also enforce performance and security standards, add a second file:

```
# .bob/rules-abap-mode/02-abap-comprehensive-standards.md

@.bob/Rules and Guardrails/rules-code-abap/abap_comprehensive_standards rule.xml
```

Step 5: Verify by activating the mode

Switch to ABAP Mode in Bob. The rules are now active — ask Bob to generate an ABAP report and confirm that it produces a proper program header, modification log, and compliant code automatically.

Rules in action — ABAP documentation example

The ABAP documentation rule loaded through `rules-abap-mode/` mandates a specific program header format. Without the rule, Bob might generate a minimal header. With the rule enforced, every generated report includes the full SAP-standard header.

Without mode-specific rule

```
** Report Z_SALES_ORDER_REPORT*

** Created by: Developer*

REPORT z_sales_order_report.

DATA lv_vbeln TYPE vbeln.
```

With mode-specific rule enforced

```
**&---------------------------------------------------------------**
**& Report Z_SALES_ORDER_REPORT*
**&---------------------------------------------------------------**
**& Change Request: CHG0012345*
**& Programmer:     John Doe*
**& Date:           2024-01-15*
**& Transport No:   DEVK900123*
**& Description:    Sales Order Processing Report*
**&---------------------------------------------------------------**
**& Modification Log:*
**& Date       | Developer  | CR Number   | Description*
**& 2024-01-15 | John Doe   | CHG0012345  | Initial Development*
**&---------------------------------------------------------------**
REPORT z_sales_order_report.
#### " Field symbol declaration -- per field symbol rule
FIELD-SYMBOLS: <fs_order> TYPE vbak.
#### " Only required fields selected -- per SELECT rule
SELECT vbeln erdat kunnr netwr
FROM vbak
INTO TABLE @DATA(lt_orders)
WHERE erdat IN s_erdat
AND loevm = space.   *" Deletion flag check -- per loevm rule*
```

> **What the rule enforces:** Program header with change-request number · modification log · field symbols for table operations · `SELECT` with explicit field list and `WHERE` clause · deletion-flag (`LOEVM`) check on `VBAK` — all injected automatically from the XML rule at mode activation.

Rules in action — UI5 project structure example

The same pattern applies to a UI5 mode. The `rules-ui5-mode/` directory loads project-structure and component-coding rules, ensuring that Bob generates files in the correct locations and follows MVC separation.

Rule files for UI5 mode

```
# .bob/rules-ui5-mode/01-ui5-project-structure.md

@.bob/Rules and Guardrails/rules-code-ui5-frontend/8_ui5_project_structure.xml
```

```
# .bob/rules-ui5-mode/02-ui5-all-frontend-rules.md

@.bob/Rules and Guardrails/rules-code-ui5-frontend/9_ui5_all_frontend_rules.xml
```

Structure-compliant output generated under UI5 mode

```
#### // webapp/controller/SalesOrder.controller.js
#### // Generated under ui5-mode with structure rules active
sap.ui.define([
"sap/ui/core/mvc/Controller",
"sap/ui/model/json/JSONModel"
], function (*Controller*, *JSONModel*) {
"use strict";
return Controller.extend("com.example.controller.SalesOrder", {
onInit: function () {
#### // Model attached to view -- not globally, per scoping rule
this.getView().setModel(new JSONModel(), "view");
},
onCreateOrder: function () {
#### // Business logic delegated to model layer -- per MVC rule
var oModel = this.getOwnerComponent().getModel();
oModel.create("/SalesOrders", oPayload, {
success: this._onSuccess.bind(this),
error:   this._onError.bind(this)
                                          });
                                        }
                              });
});
```

> **Cross-technology benefit:** A developer switching between ABAP Mode and UI5 Mode within the same project automatically receives the correct rule set — ABAP documentation standards when writing backend code, and UI5 structural rules when building the frontend — with no manual context switching required.

How *rules-{mode}/* relates to *customInstructions*

The `customInstructions` property in `custom_modes.yaml` and the `rules-{mode}/` directory are **complementary, not competing**, mechanisms. Both are loaded whenever the mode is active, and their content is merged together.

Table: Division of responsibility between `customInstructions` and `rules-{mode}/`.

Property	Best Used For	Location
customInstructions	Short behavioral directives: tone, output format, tool-calling order, and guardrails in prose	Inline, in custom_modes.yaml
rules-{mode}/ directory	Detailed external rule files: XML standards, large checklists, and rule sets shared across teams	Separate files, auto-loaded by convention

> **Attention:** There is no `ruleFiles` property in the mode YAML schema. The `rules-{mode}/` directory is the only supported mechanism for explicitly loading external rule files per mode. Do not add a custom `ruleFiles:` key to the YAML configuration — it will be silently ignored.

Governance benefits by stakeholder

- **Developers** receive consistent, automatically enforced standards for naming, structure, and documentation, removing ambiguity about what "correct" looks like on a given project.
- **Architects** gain a scalable rule architecture — global rules for organization-wide standards, domain-specific rules for shared technical policies, and mode-specific rules for per-technology enforcement — without duplicating content across configurations.
- **Project managers** gain confidence that AI-assisted output will meet the same transport, testing, and documentation requirements as manually written code, reducing rework and quality-gate failures later in the delivery cycle.
- **Senior management** gains an auditable governance layer: rules are version-controlled, owned by named teams, and enforced automatically on every interaction, supporting compliance and change-management objectives as AI-assisted development scales.

Summary and key takeaways

Mode-specific rules in IBM Bob give SAP development teams a clean, maintainable way to enforce coding standards without embedding large rule bodies inside YAML configuration.

- Create `.bob/rules-{mode-slug}/` — one directory per mode.
- Add a Markdown file per rule, using `@path/to/rule.xml` to import an external file.
- Prefix filenames numerically (`01-`, `02-`) to control load order.
- Bob loads all files in the directory automatically when the mode activates — no registration is required.
- ABAP and UI5 rule XML files remain in the master rule library under `Rules and Guardrails/` — the `rules-{mode}/` files simply reference them.

The result is a Bob configuration that scales cleanly: rules are authored once, referenced from multiple modes where needed, owned by the team that understands them, and enforced automatically every time a developer opens the relevant mode.

Intelligent Orchestration for Rules, Skills, and Code Generation for SAP (IBM Bob Mode)

A custom mode for SAP in IBM Bob would let you tailor Bob's behavior to enterprise workflows like ABAP development, RAP services, CDS Views, and integration with S/4HANA RISE. Since Bob's modes are defined in YAML, you can enforce SAP-specific rules, restrict tools, and embed prompts aligned with Clean Core and orchestration practices.

In this section we will see how IBM Bob's three-layer system — Skills, Rules, and Modes — works together to give you a purpose-built AI developer for SAP ABAP and UI5 projects. When to use each, what benefit each delivers, and exactly how to build it.

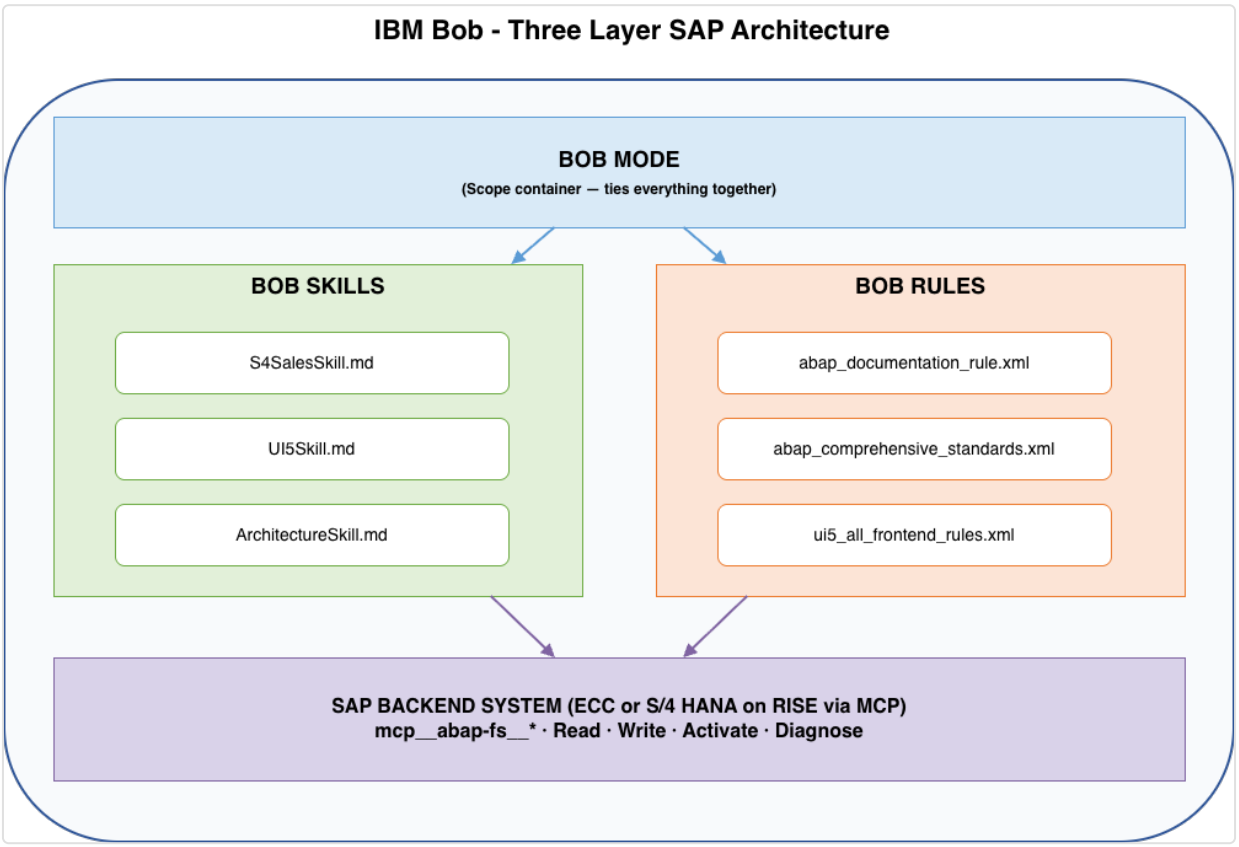


Figure: IBM Bob three-layer architecture — Skills, Rules, and Modes

The three-layer architecture

IBM Bob is not a single monolithic assistant — it is an orchestrated system. When you work on a SAP project, three distinct layers of intelligence combine to produce output that is simultaneously domain-aware, standards-compliant, and context-scoped. Understanding each layer is the key to getting the most from Bob.

Table: Three-layer architecture — Skills, Rules, and Modes

Layer	What it does	When it loads
Bob Skills	Provide domain knowledge and guidance — the "what" and "why" of SAP development	Dynamically, triggered by keywords in your prompt
Bob Rules	Enforce coding standards — the non-negotiable "how to write code correctly"	Always-on for their associated mode; loaded before every prompt
Bob Mode	Scope container that binds skills, rules, system connections, and guardrails into one purpose-built persona	On mode selection; persists for the entire session

Bob skills — Your domain expert on demand

A **Bob Skill** is a comprehensive Markdown document that packages deep domain expertise into a form Bob can consult dynamically. Think of it as a permanent subject-matter expert sitting alongside your AI developer — one who knows every SAP table, BAPI, pricing condition type, and enhancement pattern relevant to your project.

> **Critical distinction from rules:** Skills are **not** loaded before the prompt. They are loaded **dynamically** when Bob detects relevant trigger keywords in your request. This is what makes Bob efficient — it only pulls in the knowledge it actually needs for the task at hand, spending no unnecessary context on irrelevant domains.

Key characteristics of a Bob Skill

- **Comprehensive Knowledge Base**: contains deep domain expertise (tables, BAPIs, configuration, enhancements)
- **Contextual Guidance**: provides both "what" and "why" with explanations, not just code
- **Business Logic**: encodes customer-specific rules and processes beyond SAP standard
- **Code Examples**: working templates Bob can adapt, not just documentation
- **Decision Support**: helps Bob choose the right approach (BADI vs. enhancement point, BAPI vs. direct insert)

When to use a skill

Use a skill when the knowledge you are encoding has the following characteristics:

- **Domain-wide and contextual**: for example, "everything about SAP SD" or "all UI5 OData V4 patterns"
- **Advisory in nature**: guidance on which approach to use, not hard rules
- **Only relevant for certain prompts**: you do not want it consuming context when working on unrelated tasks
- **Frequently evolving**: business rules and processes change; a skill file is easy to maintain

Example: S4SalesSkill for ABAP development

The `S4SalesSkill.md` file provides everything Bob needs to build correct S/4HANA Sales and Distribution code. When a developer types "create a sales order report," Bob detects the keywords **sales order** and **report**, loads the skill, and the generated code is immediately grounded in correct table structures, authorization objects, and business rules.

Prompt → Skill trigger → Output

Table: S4SalesSkill prompt-to-output trigger example

Step	Detail
Developer says	"Create a report to show open sales orders for customer 1000"
Bob detects keywords	"sales orders", "customer" → loads S4SalesSkill
Skill provides	VBAK/VBAP structure, VBUK status, V_VBAK_VKO auth, pricing logic
Bob generates	Correct, compliant ABAP — first time, every time

Without the skill, Bob would produce generic ABAP that a developer would need to correct extensively — wrong tables, missing authorization checks, no deletion flag handling. With the skill, the output is production-ready from the first generation.

```
**&---------------------------------------------------------------------**
**& Report  Z_SD_OPEN_ORDERS_REPORT*
**& Skill: S4SalesSkill triggered by "sales orders" + "customer"*
**&---------------------------------------------------------------------**
REPORT z_sd_open_orders_report.
#### "-- Skill provides: use VBAK + VBUK for header + status join
#### "-- Skill provides: always check LOEVM (deletion flag) on VBAK
#### "-- Skill provides: authorization object V_VBAK_VKO with VKORG + ACTVT 03
AUTHORITY-CHECK OBJECT 'V_VBAK_VKO'
ID 'VKORG' FIELD p_vkorg
ID 'ACTVT' FIELD '03'.
IF sy-subrc <> 0.
MESSAGE e001(z_sd_auth) WITH p_vkorg.
ENDIF.
#### "-- Only required fields, per skill table guidance
SELECT a~vbeln a~erdat a~kunnr a~netwr b~gbstk
FROM vbak AS a
INNER JOIN vbuk AS b ON b~vbeln = a~vbeln
INTO TABLE @DATA(lt_orders)
WHERE a~kunnr = @p_kunnr
AND a~loevm = space       *"-- deletion flag: from skill*
AND b~gbstk <> 'C'.      *"-- open only (VBUK status): from skill*
```

Example: UI5 skill for frontend development

The same pattern works for UI5. A UI5 skill encodes the correct OData V4 binding patterns, `Component.js` structure, i18n conventions, and SAP Fiori UX rules. When a developer asks "create a list page for sales orders," Bob loads the UI5 skill and generates a correctly structured MVC application rather than a generic JavaScript component.

```
#### // Skill: UI5 skill triggered by "list page" + "sales orders"
#### // Skill provides: use sap.ui.model.odata.v4.ODataModel (not V2)
#### // Skill provides: binding path from skill's OData entity map
#### // Skill provides: controller stays thin — logic in model/formatter
sap.ui.define([
"sap/ui/core/mvc/Controller",
"sap/ui/model/Filter",
"sap/ui/model/FilterOperator"
], function(*Controller*, *Filter*, *FilterOperator*) {
"use strict";
return Controller.extend("com.example.sd.controller.SalesOrderList", {
onInit: function() {
#### // OData V4 model — from skill: always V4 for S/4HANA Cloud
this._oModel = this.getOwnerComponent().getModel();

},
onSearch: function(*oEvent*) {
var sQuery = oEvent.getParameter("query");
#### // Skill provides: filter on CustomerID and SalesOrderType
var aFilters = [
new Filter("SoldToParty", FilterOperator.Contains, sQuery)

  ];

this.byId("salesOrderTable").getBinding("items").filter(aFilters);

}

});
});
```

Bob rules — Non-negotiable standards, always enforced

A **Bob Rule** is a strict, always-on constraint. Unlike a skill, it is not advisory — Bob must comply with it for every piece of code it generates or reviews, regardless of what the prompt says. Rules are loaded **before** every prompt when their associated mode is active.

For SAP ABAP projects, rules typically encode corporate coding standards, transport management requirements, security mandates, and performance directives that every ABAP developer must follow. For UI5 projects, rules encode project structure conventions, component naming standards, and accessibility requirements.

Skills vs Rules — side by side

Table: Skills vs Rules comparison

Aspect	Bob Skills	Bob Rules
Purpose	Provide domain knowledge and guidance	Enforce coding standards
Scope	Broad — entire module or domain	Narrow — one specific practice
Content	"How to solve problems in this domain"	"How to write code correctly"
Flexibility	Adaptive, context-aware	Strict — must be followed
Format	Markdown documentation with examples	XML with structured rule definitions
Examples	Business processes, integration patterns, table structures	Naming conventions, header format, error handling
When used	During solution design and implementation — on demand	During every code generation and review — always
Loading mechanism	Triggered by keywords in the prompt	Loaded automatically before the prompt

What an ABAP documentation rule enforces

The `abap documentation rule.xml` is a perfect example of a rule: it defines exactly what a program header must contain, what a modification log must look like, and how data declarations must be commented. These are not suggestions — every ABAP program Bob generates must comply.

```
#### <!-- abap documentation rule.xml (excerpt) -->
<abap_documentation_rules>
<priority>CRITICAL - SAP ABAP Documentation Standards</priority>
<header_section>

<rule>
  <name>Program Header with Change Request</name>
  <requirement>Header must contain CR number, programmer,
    transport, date, description, and modification log</requirement>
</rule>

</header_section>
</abap_documentation_rules>
```

> **When to use a rule vs a skill:** If you find yourself writing "always do X" or "never do Y" — that is a rule. If you are writing "when working on SD, the relevant tables are…" — that is a skill. The distinction matters because rules consume context on every single prompt; only put constraints there that must truly apply to every output.

Bob mode — The scope container that binds everything

A **Bob Mode** is a purpose-built configuration that combines a specific SAP backend connection, a set of mandatory skills, mode-specific rules, guardrails, and custom instructions into a single selectable persona. A developer selects a mode once at the start of a session, and Bob operates entirely within that scope for the duration — never deviating, never pulling in irrelevant skills, never dropping a coding standard.

This is particularly powerful in SAP environments where the same developer might work on ABAP backend code in the morning and a UI5 Fiori app in the afternoon. Two modes means two completely different, correctly-scoped AI developers — one connected to the ABAP backend system `FS3` with SD expertise, and one focused on frontend UI5 component structure.

> **Efficiency benefit:** Because each mode pre-loads only the rules and instructs Bob to load only the relevant skills, you spend Bob context budget efficiently. An ABAP mode session never wastes context on UI5 frontend rules. A UI5 mode session never loads SD table structures. Your results are more focused and your token costs are lower.

What a mode definition contains

- `slug`: unique identifier, also drives the `rules-{slug}/` directory convention
- `roleDefinition`: Bob's identity and expertise for this scope
- **SAP backend binding**: which MCP connection (e.g. FS3 via `mcp__abap-fs__*`) to use

- **Skills to load**: which skill files to consult for this domain
- `customInstructions`: session-level guardrails (never use `SELECT *`, never `COMMIT` in enhancements)
- `rules-{slug}/` **directory**: external rule files auto-loaded at every prompt

How to build this for an SAP project — Step by step

Here is the complete setup for a typical SAP S/4HANA project with both an ABAP backend developer and a UI5 frontend developer using Bob.

Target project structure

```
.bob/

├── custom_modes.yaml
├── BOB-Skills/
│   ├── S4SalesSkill.md
│   ├── ABAP/
│   │   ├── ABAP-STANDARD-INTEGRATION.md
│   │   └── ARCHITECTURE.md
│   └── sap-abap-outside/md/
├── Rules and Guardrails/
│   ├── rules-code-abap/
│   │   ├── abap documentation rule.xml
│   │   └── abap_comprehensive_standards rule.xml
│   └── rules-code-ui5-frontend/
│       ├── 8_ui5_project_structure.xml
│       └── 9_ui5_all_frontend_rules.xml
├── rules-abap-mode/
│   ├── 01-abap-documentation-rule.md
│   └── 02-abap-comprehensive-standards.md
└── rules-ui5-mode/
├── 01-ui5-project-structure.md
└── 02-ui5-frontend-rules.md
```

Step 1: Author the skills

Create `.bob/BOB-Skills/S4SalesSkill.md`. The file must include trigger keywords in its metadata so Bob knows when to auto-load it:

```
# SAP S/4HANA Sales Module — Bob Skill

## Skill Metadata

- **Skill Name**: S4SalesSkill

- **Trigger Keywords**: sales, order, quotation, delivery, billing, pricing, customer, SD module, VBAK, VB
```

Sales Document Structure

Key Tables

- **VBAK**: Sales Document Header
- **VBAP**: Sales Document Item
- **VBUK**: Sales Document Header Status (always join for open orders)
- **VBFA**: Sales Document Flow (for document chain tracing)

- **KONV**: Pricing Conditions (for pricing BADIs)

Authorization Objects

- **V_VBAK_VKO**: Sales order display/create — always check VKORG + ACTVT

Standard BAPIs

- `BAPI_SALESORDER_CREATEFROMDAT2`: create sales orders
- `BAPI_SALESORDER_CHANGE`: change existing orders
- `BAPI_SALESORDER_GETLIST`: list orders by criteria

Customer-Specific Business Rules

- All orders above EUR 50,000 require dual approval (check `ZZVBAK-APPR_STATUS`)
- Volume discount threshold: 100 units triggers `ZK007` condition type

Step 2: Author the rule XML files

Place XML rule files under `.bob/Rules and Guardrails/rules-code-abap/`. Each XML file should carry a `<priority>CRITICAL</priority>` tag so Bob treats it as non-negotiable.

Step 3: Create the *rules-{mode}/* directories

Create `.bob/rules-abap-mode/` and populate it with small Markdown files that use the `@file` import syntax to reference the XML rules:

```
# .bob/rules-abap-mode/01-abap-documentation-rule.md

@.bob/Rules and Guardrails/rules-code-abap/abap documentation rule.xml
```

```
# .bob/rules-abap-mode/02-abap-comprehensive-standards.md

@.bob/Rules and Guardrails/rules-code-abap/abap_comprehensive_standards rule.xml
```

> **Note:** Files are loaded alphabetically — prefix with numbers to control order.

Step 4: Define the mode in *custom_modes.yaml*

Create an entry that binds the SAP backend, specifies which skills to load, and adds session guardrails:

```
customModes:

- slug: abap-mode

name: ABAP Mode

roleDefinition: >

 You are an expert SAP ABAP developer for S/4HANA.

  ## SAP Backend — MANDATORY

  Use mcp__abap-fs__* tools with connectionId: "FS3" for all

  object reads, writes, and activations. Call

  mcp__abap-fs__get_connected_systems first.

  ## Skills — MANDATORY

  Before any ABAP response, load:

  - .bob/BOB-Skills/S4SalesSkill.md

  - .bob/BOB-Skills/ABAP/ABAP-STANDARD-INTEGRATION.md

customInstructions: >

Rules loaded from .bob/rules-abap-mode/ apply to every output.

 Never SELECT *. Never COMMIT WORK in enhancements.

 Never save to $TMP. Always remove breakpoints before transport.


groups:

 - read

 - edit

 - execute

 - mcp
```

Step 5: Repeat for the UI5 mode

Create `slug: ui5-mode` in the YAML with its own `roleDefinition` focused on SAPUI5/OData V4, and populate `.bob/rules-ui5-mode/` with files importing the UI5 XML rules.

Step 6: Select the mode and start coding

In Bob's sidebar, select **ABAP Mode**. Bob will confirm the FS3 backend connection, pre-load the rules, and stand ready to auto-invoke skills from any SD-related keyword in your prompt.

Example — Pricing BADI in ABAP mode

A developer types: **"Implement volume discount for sales orders above 100 units using a BADI"**

Here is what the three layers contribute in sequence:

Table: Three-layer contribution to a single IBM Bob response

Layer	Contribution to the response
Rules (pre-loaded)	Mandates: complete program header with CR number · no hard-coded threshold — read from TVARVC · no COMMIT WORK inside the BADI · authorization check on V_VBAK_VKO
S4SalesSkill (auto-loaded on "pricing", "discount", "orders")	Provides: BADI_SD_PRICING is the correct enhancement · ZK007 is the custom condition type · KONV table structure · pricing procedure flow · customer rule — threshold is 100 units
Mode (FS3 backend)	Verifies: reads actual BADI_SD_PRICING definition from FS3 via mcp__abap-fs__search_abap_objects · confirms TVARVC entry name · confirms ZK007 exists in system

```
**&---------------------------------------------------------------------**
**& BADI Implementation: BADI_SD_PRICING / ZCL_SD_PRICING_IMPL*
**& Change Request: CHG0012345*
**& Programmer:     Jane Smith*
**& Date:           2024-06-01*
**& Transport:      DEVK900456*
**& Description:    Volume discount — orders above 100 units (ZK007)*
**&---------------------------------------------------------------------**
**& Modification Log:*
**& Date       | Developer  | CR No      | Description*
**& 2024-06-01 | Jane Smith | CHG0012345 | Initial Development*
**&---------------------------------------------------------------------**
CLASS zcl_sd_pricing_impl DEFINITION PUBLIC

                        FINAL CREATE PUBLIC.

PUBLIC SECTION.
INTERFACES if_badi_sd_pricing.
ENDCLASS.
CLASS zcl_sd_pricing_impl IMPLEMENTATION.
METHOD if_badi_sd_pricing~calculate.
DATA: lv_threshold TYPE decfloat34,

      lv_discount  TYPE kbetr.

#### "-- No hard-coded threshold — read from TVARVC (rule: no hard-coding)
SELECT SINGLE low FROM tvarvc
INTO lv_threshold
WHERE name = 'Z_VOL_DISC_THRESHOLD'
AND  type = 'P'.
#### "-- Skill: check item quantity against threshold
CHECK im_kwmeng >= lv_threshold.
#### "-- Skill: write to ZK007 condition value (custom volume discount)
```

```
LOOP AT ct_komv ASSIGNING FIELD-SYMBOL(<fs_kondm>)  *"field symbol: rule*

WHERE kschl = 'ZK007'.

<fs_kondm>-kbetr = '500'.   *"-- 5% in per-mille*

ENDLOOP.

#### "-- Rule: NEVER call COMMIT WORK inside a BADI

#### "-- Rule: Remove all breakpoints before transport

ENDMETHOD.

ENDCLASS.
```

Every line traced to a layer:

- Program header format → documentation rule
- No hard-coded threshold → comprehensive standards rule
- TVARVC lookup → ABAP standards rule
- `BADI_SD_PRICING` + `ZK007` + KONV → S4SalesSkill
- `FIELD-SYMBOL` for loop → documentation rule
- No `COMMIT WORK` → guardrail in mode `customInstructions`

Benefits at a glance

Table: Benefits of Skills, Rules, and Modes for ABAP and UI5 development

Benefit	ABAP Development	UI5 Development
Consistency	Every report, BADI, and class follows the same header format and naming standard — across all developers	Every controller, view, and fragment follows the same MVC structure and OData binding pattern
Speed	Bob knows VBAK/VBAP/VBUK, correct BAPIs, and auth objects — no developer has to look them up	Bob knows Fiori layout patterns, SAP icon library, and i18n conventions immediately
Correctness	Authorization checks, deletion flags, and lock concepts are present in first generation	Component.js lifecycle, manifest.json structure, and OData V4 bindings are correct from the start
Onboarding	New ABAP developers produce standards-compliant code on their first day in the mode	Frontend developers unfamiliar with SAP get correct UI5 structure guided by the skill
Token efficiency	ABAP mode never loads UI5 rules; context budget spent only on what is relevant	UI5 mode never loads SD table structures; every token goes to frontend knowledge

Decision guide — What goes where

Table: Decision guide — what to encode as a Skill, Rule, or Mode

You want to encode…	Use
All SAP SD tables, BAPIs, and enhancement points	Skill (S4SalesSkill.md)
Customer-specific pricing rules and business logic	Skill (business logic section)
Mandatory program header format with CR number	Rule (abap documentation rule.xml)
Never SELECT *, always check LOEVM	Rule (abap_comprehensive_standards rule.xml)
SAP system to connect and connection ID to use	Mode (roleDefinition + MCP config)
Which skills are relevant for this domain	Mode (roleDefinition — skills list)
UI5 project folder structure and naming	Rule (ui5_project_structure.xml)
Fiori UX patterns and OData V4 binding approaches	Skill (UI5 skill)
Switching between ABAP and UI5 development contexts	Mode (select ABAP Mode or UI5 Mode)

Key takeaways

The real power of IBM Bob for SAP projects is not any single capability — it is the **orchestration** of all three layers working together. Rules guarantee standards. Skills deliver domain intelligence. Modes scope the entire session so Bob is always purpose-built for the task at hand.

For an S/4HANA engagement, this means: an ABAP developer selects ABAP Mode, asks for a pricing BADI, and receives production-ready code — correct tables, correct authorization, correct documentation format, correct BADI implementation pattern — without any back-and-forth corrections. A UI5 developer selects UI5 Mode, asks for a list page, and receives a correctly structured MVC application with proper OData V4 bindings and Fiori UX patterns.

The investment is building the skills and rules once. The return is every developer on the team — regardless of SAP experience level — producing consistently high-quality, standards-compliant code from the first prompt.

Centralized IBM Bob configuration and distribution for SAP teams

This section documents a governance architecture for distributing IBM Bob configuration across a multi-user, multi-business-area SAP development team through a central Git repository. It describes a five-layer model — Top Governance, Skill, Connectivity, Environment Variable, and User — that separates what belongs in shared infrastructure from what belongs on the developer's machine, and explains how to build, maintain, and consume it. The section is written for a mixed audience of SAP developers, solution architects, project managers, and senior management.

A SAP development program that deploys IBM Bob across more than one developer immediately faces a configuration problem. Each developer needs the same coding standards, the same domain skills, and the same system connections. Each developer also has credentials, preferences, and project-specific context that should never leave their own machine. Solving this through individual, per-developer setup is a maintenance failure waiting to happen: one developer receives the ABAP documentation rule and another does not; someone updates the SD pricing skill and the rest of the team never sees the change.

The answer is a centralized Git repository that acts as the authoritative source for every configuration artifact the team shares. Developers clone it once and pull updates as they arrive, and their local IBM Bob workspace inherits the

shared baseline automatically. What must stay local — credentials, personal preferences, and user-specific skills — lives in the developer's own IBM Bob configuration and never touches the shared repository.

This section specifies the five-layer model that makes this possible, explains what belongs in each layer and why, and provides the directory structure, file templates, and consumption procedure a team needs to stand it up.

Scope

This section covers the design and operation of a central configuration repository for IBM Bob across a SAP delivery program: the five-layer governance model, the canonical repository structure, the setup procedure for a new developer, the update and promotion workflows, and the security properties the model is designed to guarantee. It assumes familiarity with IBM Bob's configuration mechanisms (modes, rules, skills, and MCP server definitions) as introduced elsewhere in this publication.

The five-layer model

The central configuration repository is organized as five layers, each with a distinct governance boundary. The separation is not cosmetic: it reflects a real difference in who owns the content, how often it changes, and what the security consequences of leaking it would be.

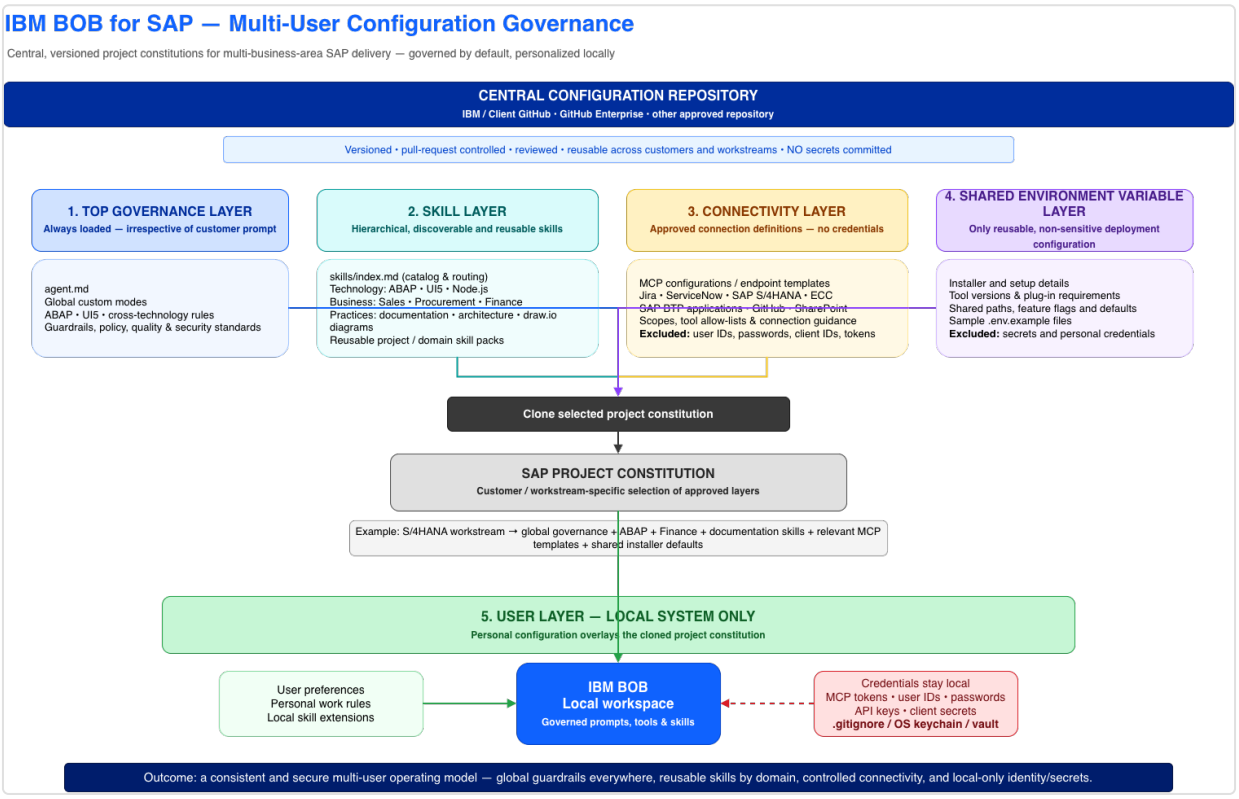


Figure: IBM Bob five-layer multi-user configuration governance model for SAP delivery

Each layer loads independently:

- **Top Governance Layer**: loads unconditionally, for every developer, on every session.
- **Skill Layer**: loads dynamically, when trigger keywords relevant to a skill appear in a prompt.
- **Connectivity Layer**: defines how IBM Bob reaches SAP and non-SAP systems, without encoding the secrets that authenticate to them.
- **Environment Variable Layer**: carries the non-secret configuration values that an installer or workspace needs.
- **User Layer**: exists only on the developer's local machine and is never pushed to the repository.

The Layer 1 — Top Governance Layer through Layer 5 — User layer sections examine each layer in turn. The Repository Structure section first sets out the canonical directory structure that hosts layers 1 through 4 in the shared repository.

Repository structure

The canonical directory layout for the central repository is as follows:

```
bob-config/

├── README.md
├── LAYER-INDEX.md         ← Master index of all layers
│
├── 01-governance/         ← Top Governance Layer
│   ├── agent.md           ← Global agent behaviour
│   ├── modes/
│   │   ├── abap-mode.yaml
│   │   ├── ui5-mode.yaml
│   │   ├── cap-nodejs-mode.yaml
│   │   └── architecture-mode.yaml
│   └── rules/
│       ├── abap/
│       │   ├── abap-documentation-rule.xml
│       │   └── abap-comprehensive-standards.xml
│       ├── ui5/
│       │   ├── ui5-project-structure.xml
│       │   └── ui5-frontend-standards.xml
│       └── cap/
│           ├── cap-model-cds.xml
│           └── cap-service-logic.xml
│
├── 02-skills/               ← Skill Layer
│   ├── SKILL-INDEX.md       ← Master skill index
│   ├── technology/
│   │   ├── abap/
│   │   │   ├── abap-core.md
│   │   │   ├── abap-rap.md
│   │   │   └── abap-cds-views.md
│   │   ├── nodejs/
│   │   │   ├── cap-nodejs-core.md
│   │   │   └── cap-nodejs-odata.md
│   │   └── ui5/
│   │       ├── ui5-fiori-elements.md
│   │       └── ui5-freestyle.md
│   ├── business-area/
│   │   ├── sales/
│   │   │   └── s4-sales-sd.md
│   │   ├── procurement/
│   │   │   └── s4-procurement-mm.md
│   │   └── finance/
│   │       └── s4-finance-fi-co.md
│   └── documentation/
│       ├── architecture-drawio.md
│       └── technical-spec.md
│
├── 03-connectivity/          ← Connectivity Layer
│   ├── mcp-servers.json      ← Central MCP server definitions
│   └── system-catalog.md     ← System inventory
│
├── 04-environment/           ← Environment Variable Layer
│   └── installer-config.env.template
│
└── 05-user-template/         ← User Layer reference only
    ├── README.md
    └── mcp.json.template
```

The `05-user-template/` directory ships reference files and instructions only. It contains no real credentials, and it is the only user-facing content that belongs in the repository — its purpose is to tell developers exactly what to configure locally, not to store any of it.

Layer 1 — Top governance layer

Purpose and contents

The Top Governance Layer holds every configuration artifact that must load unconditionally for every developer, regardless of which prompt they submit, which project they are working on, or which SAP system they are connected to. It is the foundation that guarantees a consistent baseline across the team.

The Top Governance Layer contains the following three categories of content:

- `agent.md` **— global agent behavior.** This file defines the non-negotiable operating principles that IBM Bob applies in every session. It is not a skills document and not a coding standard — it is the behavioral contract. A typical example is a directive such as: SAP System: `FS3` (S/4HANA). Always pass `connectionId: "FS3"` in `abap-fs` calls. Transport rule: never use `$TMP`; use Z-packages only.
- **Global custom modes.** The mode YAML files in `01-governance/modes/` define the IBM Bob modes available to every developer. Each mode binds a role definition, a set of skills to consult, an SAP system connection, and session-level guardrails. Modes are the primary scope container: they determine which rules load, which skills are pre-loaded in the role definition, and which backend system IBM Bob connects to.
- **Rules by technology.** The XML rule files in `01-governance/rules/` are the coding standards that apply to every output IBM Bob generates within their associated mode. ABAP rules enforce documentation headers, naming conventions, transport handling, and security mandates. UI5 rules enforce project structure, OData binding patterns, and accessibility requirements. CAP rules enforce CDS model conventions and service logic structure. These files load before every prompt when their associated mode is active.

The agent.md file

A project-level `agent.md` lives in the root of the central repository and is copied (or symlinked) into each developer's `.bob/` directory during setup. A minimal template:

```
# Global Agent Directives — SAP Programme [Programme Name]

## SAP System Binding

- Primary backend: FS3 (S/4HANA)

- Always pass connectionId: "FS3" in all abap-fs MCP calls

- Transport rule: NEVER use $TMP. All development in Z-packages only

- Documentation rule: all ABAP objects must include the standard header

  (Program, Developer, CR Number, Transport)

## Cross-Mode Standards

- Never generate code that requires manual correction for standards compliance

- Always surface the CR number and transport in generated headers

- Never commit credentials to any file that might be pushed to version control
```

Maintaining governance artifacts

The Top Governance Layer is owned by the SAP technical lead or architect, not by individual developers. Changes to rules and modes go through a pull-request review process; developers consume the layer by pulling the repository, and do not modify its contents directly.

This governance boundary is the most important one in the model. A rule that is wrong in `agent.md` is wrong for everyone. A mode that points to the wrong SAP system connection makes every developer's session unreliable. The review gate exists precisely to prevent silent breakage at the foundation.

Layer 2 — Skill layer

Purpose and structure

Skills are the team's collective domain intelligence, packaged as Markdown files that IBM Bob loads dynamically when relevant trigger keywords appear in a prompt. A developer working on a purchase order report does not need the Finance skills loaded; one working on a CO settlement run does not need SD pricing logic. The Skill Layer is organized so that IBM Bob pulls exactly the knowledge it needs, and nothing more.

The hierarchy has three branches:

- **Technology skills**: runtime-level expertise that applies regardless of business area. ABAP core, ABAP RAP, CDS views, Node.js CAP, UI5 Fiori Elements, and UI5 freestyle are all technology skills.
- **Business area skills**: domain-level expertise anchored to a specific SAP module. Sales and Distribution, Materials Management, and Financial Accounting are business area skills; they contain the tables, BAPIs, authorization objects, enhancement points, and customer-specific business rules for their module.
- **Documentation skills**: expertise in producing artifacts that are not code. Architecture diagrams in draw.io format, technical specification documents, and functional design documents are documentation skills.

The Skill index file

The `SKILL-INDEX.md` file at the root of the skills directory is the team's authoritative catalog of what exists and how to trigger it. Every skill added to the repository must be registered here. The index is what IBM Bob — and human developers — consult to understand what domain intelligence is available.

```
# Skill Index — SAP Programme [Programme Name]

Last updated: [date] | Maintained by: [owner]

## Technology Skills

**Table: Technology skills — trigger keywords and coverage**

| Index | Skill file | Trigger keywords | Coverage |
|-------|-----------|------------------|----------|
| T-01 | technology/abap/abap-core.md | ABAP, internal table, SELECT, class, method, BADI | Core ABAP lang
| T-02 | technology/abap/abap-rap.md | RAP, BDEF, managed BO, unmanaged BO, EML | ABAP RESTful Application
| T-03 | technology/abap/abap-cds-views.md | CDS, annotation, association, projection view | CDS view patt
| T-04 | technology/nodejs/cap-nodejs-core.md | CAP, CDS, Node.js, service handler, srv.on | CAP Node.js s
| T-05 | technology/ui5/ui5-fiori-elements.md | Fiori Elements, list report, object page | Fiori Elements
| T-06 | technology/ui5/ui5-freestyle.md | controller, view, component, manifest, fragment | SAPUI5 freest

## Business Area Skills

**Table: Business area skills — trigger keywords and coverage**

| Index | Skill file | Trigger keywords | Coverage |
|-------|-----------|------------------|----------|
| B-01 | business-area/sales/s4-sales-sd.md | sales order, quotation, delivery, pricing, SD | S/4HANA Sale
| B-02 | business-area/procurement/s4-procurement-mm.md | purchase order, goods receipt, MM, EKKO | S/4HAN
| B-03 | business-area/finance/s4-finance-fi-co.md | journal entry, cost center, settlement, FI, CO | S/4H

## Documentation Skills

**Table: Documentation skills — trigger keywords and coverage**

| Index | Skill file | Trigger keywords | Coverage |
|-------|-----------|------------------|----------|
| D-01 | documentation/architecture-drawio.md | architecture diagram, draw.io, sequence diagram | Architec
| D-02 | documentation/technical-spec.md | technical specification, functional spec | Technical and functi
```

The index enforces two disciplines. First, it prevents duplication: before adding a skill, a developer checks the index and discovers that the knowledge they want already exists. Second, it makes trigger-keyword management explicit — overlapping triggers between skills cause IBM Bob to load both when only one is relevant, which wastes context budget. The index makes those overlaps visible before they cause problems.

Authoring business area skills

A business area skill is the most technically dense artifact in the layer. It must contain enough information that IBM Bob can generate correct, production-ready code for the module without the developer needing to provide table names, BAPI signatures, or authorization objects. The standard structure for a business area skill is:

```
# [Module Name] — Bob Skill

# SAP Programme [Name] | Business Area: [Area] | Module: [SD / MM / FI / CO]

## Skill Metadata

- Trigger keywords: [comma-separated list]

- SAP system: S/4HANA [release]

- Maintained by: [owner name] | Last updated: [date]

## Document / Object Structure

### Key Tables

- [TABLE]: [description] — [notes on when to use]

### Status and Deletion Flags

- [specific flags and their meaning in this module]

## Authorization Objects

- [AUTH_OBJECT]: [description] — always check [field] with [activity]

## Standard BAPIs and Function Modules

- [BAPI_NAME]: [purpose] — [key parameters]

## Enhancement Points and BADIs

- [BADI_NAME]: [purpose] — [when to implement]

## Customer-Specific Business Rules

- [Rule description] — [table or field reference where applicable]
```

Business area skills are owned by the module lead — the senior developer who knows the SAP module best. They are not general documentation; they are executable knowledge that IBM Bob acts on directly. A skill that contains an incorrect table name produces incorrect code in production. The review standard for skill content should be the same as the review standard applied to production code.

Layer 3 — Connectivity layer

What belongs here — and what does not

The Connectivity Layer holds the structural definition of every MCP server the team uses: where it is, which transport protocol it speaks, and which tools it exposes. What it does not hold, under any circumstances, is a credential — no client secret, no password, no OAuth token, and no API key.

> **Attention:** The most important boundary in this model
>
> A `mcp.json` file that contains a `client_secret` field is a vulnerability the moment it is committed to any repository, even a private one. The Connectivity Layer encodes the shape of the connection; the User Layer supplies the credentials that complete it.

The mcp-servers.json file

The central `mcp-servers.json` defines every MCP server the team connects to, with placeholder values for any field that will hold a secret:

```
{
"mcpServers": {
"abap-fs": {
"url": "http://localhost:4847/mcp",
"headers": {
"Authorization": "Bearer {{ABAP_FS_TOKEN}}"
      },
"timeout": 120000,
"alwaysAllow": [
"search_abap_objects",
"get_abap_object_lines",
"create_object_programmatically",
"abap_activate",
"get_abap_diagnostics"
      ],
"_comment": "ABAP FS server — token supplied via User Layer env variable ABAP_FS_TOKEN"
    },
"BTP Administration": {
"type": "streamable-http",
"url": "https://sso.mcp.btp.cloud.sap/mcp",
"oauth": {
"clientId": "e789ba01-5612-47ee-bfe7-79e26411c1ca"
      },
"_comment": "OAuth clientId is public — token obtained interactively via browser SSO"
    },
"jira": {
"type": "streamable-http",
"url": "https://your-jira-instance.atlassian.net/mcp",
"headers": {
"Authorization": "Bearer {{JIRA_API_TOKEN}}"
      },
```

```
"_comment": "JIRA API token supplied via User Layer env variable JIRA_API_TOKEN"

    },

"servicenow": {

"type": "streamable-http",

"url": "https://{{SN_INSTANCE}}.service-now.com/sncapps/mcp-server/mcp/sn_mcp_server_default",

"headers": {

"Authorization": "Bearer {{SN_ACCESS_TOKEN}}"

      },

"_comment": "ServiceNow instance name and token supplied via User Layer"

    }

  }

}
```

The `{{PLACEHOLDER}}` convention signals to every developer reading the file that this value is not stored here. It is a deliberate contract: the repository defines the topology of the integration, and the developer's local environment supplies the authentication.

The system catalog

The `system-catalog.md` file is a prose inventory of every SAP and non-SAP system the team connects to. It records the system ID, the landscape it belongs to, the type of connection used, and the team responsible for it. Developers consult it when configuring their User Layer credentials.

SAP systems

Table: SAP system connections — landscape, MCP server, and auth type

System ID	Description	Landscape	MCP Server	Auth type
FS3	S/4HANA Development	Development	abap-fs	Bearer token (ABAP FS proxy)
FS4	S/4HANA Quality	Quality	abap-fs	Bearer token (ABAP FS proxy)
BTP-DEV	BTP Development subaccount	Development	BTP Administration	OAuth 2.0 SSO
BTP-PRD	BTP Production subaccount	Production	BTP Administration	OAuth 2.0 SSO

Non-SAP systems

Table: Non-SAP system connections — MCP server and auth type

System	Description	MCP Server	Auth type
Jira	Project tracking	jira	Personal API token
ServiceNow	ITSM	servicenow	OAuth access token

Layer 4 — Environment variable layer

What belongs here

The Environment Variable Layer holds the non-secret configuration values that need to be shared across the team: installer version numbers, Node.js version requirements, npm registry URLs for internal packages, proxy settings for corporate network environments, and any other value that is the same for every developer on the project and carries no security risk if exposed.

> **Important:** The test for whether something belongs here is simple: could a competitor read this file and use the information to cause harm? If the answer is no, it belongs in the Environment Variable Layer. If the answer is yes, it belongs in the User Layer.

The installer config template

The repository ships an `installer-config.env.template` file that developers copy locally, rename to `installer-config.env`, and populate. The template contains only non-secret values already filled in, and commented placeholders for anything the developer must supply themselves:

```
#### # installer-config.env.template

#### # Copy to installer-config.env in your local bob workspace.

#### # Fill in the USER LAYER VARIABLES section with your personal credentials.

#### # Never commit installer-config.env to version control.

#### # ── SHARED VARIABLES (same for all developers) ──────────────────

BOB_SHELL_VERSION=2.0.5

NODEJS_MIN_VERSION=22.15.0

NPM_REGISTRY=https://registry.npmjs.org/

INTERNAL_NPM_REGISTRY=https://npm.your-company-registry.example.com/

#### # BAS Target URL (non-secret — identifies the environment, not the user)

BAS_URL=https://your-trial-subdomain.applicationstudio.cloud.sap

#### # ── USER LAYER VARIABLES (fill in locally — never commit) ────────

#### # BTP_EMAIL=<your BTP email>

#### # BTP_PASSWORD=<your BTP password — or use BAS_ACCESS_TOKEN instead>

#### # BAS_ACCESS_TOKEN=<your OAuth 2.0 token>

#### # ABAP_FS_TOKEN=<your ABAP FS bearer token>

#### # JIRA_API_TOKEN=<your Jira personal access token>

#### # SN_ACCESS_TOKEN=<your ServiceNow access token>
```

The template itself is committed to the repository. The populated `installer-config.env` file is always listed in `.gitignore`.

Layer 5 — User layer

What stays local

The User Layer is everything that belongs to the individual developer and must never enter the repository. Its contents fall into three categories:

- **Credentials.** Every bearer token, OAuth access token, API key, password, and client secret. The Connectivity Layer names these with `{{PLACEHOLDER}}` conventions; the User Layer is where those placeholders are resolved with real values, in the developer's own `~/.bob/settings/mcp.json` or local environment file.
- **Personal preferences and rules.** Conventions the developer follows that are not team standards — a preference for a particular commenting style, personal shorthand instructions, or project-specific context that applies only to one developer's current task. These belong in a local rule file or a local section of `agent.md`, not in the shared repository.
- **User-specific skills.** Skills that encode knowledge relevant only to a single developer's current assignment. A developer working on a niche integration who has built a skill specific to that integration's message structure should keep that skill local until it is generalizable enough to merit promotion into the shared Skill Layer.

Local configuration structure

After cloning the central repository, a developer's local IBM Bob workspace has two configuration roots: the cloned repository (shared, read from) and the local `.bob/` directory (personal, written to):

```
~/
├── .bob/
│   ├── settings/
│   │   └── mcp.json          ← Personal copy — contains real credentials
│   ├── rules/
│   │   └── user-personal-rules.md ← Personal coding preferences
│   └── skills/
│       └── [user-specific skills] ← Skills not yet promoted to shared repo
│
└── Documents/IBM BOB/Workspace/
    ├── bob-config/               ← Cloned central repository (read-only for developers)
    │   ├── 01-governance/
    │   ├── 02-skills/
    │   ├── 03-connectivity/
    │   ├── 04-environment/
    │   └── 05-user-template/
    └── [project repositories]
```

The `~/.bob/settings/mcp.json` file is the developer's personal MCP configuration. It starts from the template in `03-connectivity/mcp-servers.json`, with every `{{PLACEHOLDER}}` value replaced by the developer's real credentials. It lives in the home directory, outside any repository, and is never committed anywhere.

Setting up a new developer

A new developer joining the program completes setup in four steps. The entire process takes less than thirty minutes.

Step 1: Clone the central repository.

```
cd "/Users/<your-username>/Documents/IBM BOB/Workspace"

git clone https://github.com/<org>/bob-config.git
```

Step 2: Copy the installer config template and fill in personal credentials.

```
cp bob-config/04-environment/installer-config.env.template \

   installer-config.env

#### # Edit installer-config.env and fill in your personal credentials

#### # (BTP_EMAIL, BTP_PASSWORD, ABAP_FS_TOKEN, etc.)
```

Step 3: Build a personal mcp.json from the connectivity template.

```
cp bob-config/03-connectivity/mcp-servers.json ~/.bob/settings/mcp.json

#### # Open ~/.bob/settings/mcp.json and replace all {{PLACEHOLDER}} values

#### # with credentials from installer-config.env
```

Step 4: Verify the connection

Open IBM Bob and select the appropriate mode from `bob-config/01-governance/modes/`. Submit a verification prompt — for example, "What ABAP objects exist in package ZSALES?" for the ABAP mode. If the mode loads correctly and the MCP connection responds, setup is complete.

> **Note:** The cloned `bob-config/` directory is read-only for developers. It is the shared repository; changes to governance artifacts, skills, and connectivity definitions are proposed through pull requests and reviewed by the technical lead before merging.

Consuming Updates

When the technical lead publishes an update to the central repository — a new skill, a revised rule, an additional SAP system in the connectivity catalog — every developer picks it up with a single pull:

```
cd "/Users/<your-username>/Documents/IBM BOB/Workspace/bob-config"

git pull origin main
```

IBM Bob reads the configuration files at session start. After pulling, the developer restarts their IBM Bob session and the updated configuration is active immediately — no manual file copying, no manual rule registration, and no risk of one developer running a different version than another.

This is the core value proposition of the centralized model. The team's collective domain intelligence — every skill, every rule, every system connection definition — converges on a single version at pull time. Divergence between developers becomes a merge conflict in Git, visible and resolvable, rather than a silent inconsistency that produces different code quality from different machines.

Promotion workflow — From personal to shared

Skills and rules start locally. When a developer builds a skill that proves useful beyond their immediate assignment — a new business area skill, a rule that catches a recurring class of error, a mode variant that works well for a specific project type — it should be promoted into the shared repository by using the following steps:

1. The developer places the new artifact in the appropriate layer directory of their local clone.
2. They update `SKILL-INDEX.md` if the artifact is a skill, or the relevant mode YAML if it is a rule referenced by a mode.
3. They open a pull request against the central repository, describing what the artifact covers and why it belongs in the shared layer rather than the User Layer.
4. The technical lead reviews the content for accuracy, trigger-keyword conflicts, and standards compliance. For business area skills, the module lead is also a required reviewer.
5. On merge, every developer on the team gains the artifact on their next `git pull`.

This workflow keeps the shared layers curated. Not everything a developer builds is ready for the team; the promotion gate ensures that what enters the shared repository is accurate, maintainable, and genuinely useful to more than one person.

Multi-business-area operation

The five-layer model scales naturally to programs with teams across multiple SAP modules — SD, MM, FI, CO, PP, and others — each with different domain expertise and different SAP system access requirements.

Every layer is additive, not exclusive. A developer working on SD and FI simultaneously has both business area skills available: the ABAP mode's role definition lists both, and IBM Bob loads whichever skill's trigger keywords appear in the prompt. The Connectivity Layer contains entries for both the SD team's primary system and the FI team's system, and the developer's User Layer credentials grant access to both.

The governance boundary remains intact: the technical lead controls what enters the shared layers, and each business area module lead is the content authority for their own business area skill files. Cross-module rules — documentation standards, transport handling, security mandates — live in the Top Governance Layer and apply to everyone.

Table: Layer ownership across business area teams (Demo)

Layer	SD team	MM team	FI/CO team	Shared
Top Governance	✓ same	✓ same	✓ same	✓ all three
Skills (technology)	✓ same	✓ same	✓ same	✓ all three
Skills (business area)	SD skill	MM skill	FI/CO skill	Each team owns its module
Connectivity	FS3 (SD)	FS3 (MM)	FS3 (FI)	Shared connectivity catalog
Environment	✓ same	✓ same	✓ same	✓ all three
User Layer	Personal	Personal	Personal	Never shared

Security summary

The five-layer model contains security properties that are worth stating explicitly, because they explain why the layer separation is structured the way it is.

- **No credential enters the repository.** The Connectivity Layer uses `{{PLACEHOLDER}}` syntax for every value that would authenticate a user or service. The Environment Variable Layer's template file is committed; the populated file is listed in `.gitignore`. The User Layer never touches the repository. A repository compromise exposes topology and domain intelligence, but not a single working credential.
- **The repository is additive, not destructive.** Pulling a repository update can add a new skill or tighten a rule, but it cannot reach into a developer's local credentials or personal preferences. The User Layer is structurally isolated from the pull mechanism.
- **Credential rotation affects only the User Layer.** When a BTP password, OAuth token, or ABAP FS bearer token rotates, only the individual developer's `~/.bob/settings/mcp.json` needs updating. The central repository is unchanged, and there is no coordination overhead for credential rotation.
- **The governance boundary enforces review.** Changes to rules and modes — the artifacts that most directly affect the code IBM Bob generates — require a pull-request review before they reach any developer. A rule that is wrong, or a mode that points to the wrong system, cannot be silently deployed to the team.

Summary and key takeaways

Distributing IBM Bob configuration through a central Git repository transforms a per-developer setup problem into a managed, versioned, auditable team capability. The five-layer model provides the structure that makes it work.

The Top Governance Layer guarantees a consistent behavioral baseline. The Skill Layer gives every developer access to the team's accumulated domain intelligence. The Connectivity Layer defines the integration topology without exposing credentials. The Environment Variable Layer distributes the non-secret configuration values that differ from the defaults. And the User Layer keeps credentials and personal preferences exactly where they belong — local, secure, and never at risk of accidental exposure.

Three operating principles govern the model in production:

- **The shared repository is the source of truth for shared knowledge.** When a skill is wrong or a rule is missing, the fix goes into the repository and reaches every developer through a pull. Per-developer workarounds that bypass this path create the divergence the model exists to prevent.
- **Credentials never cross the repository boundary.** Not in comments, not in example files, not in templates with real values. The `{{PLACEHOLDER}}` convention is a hard rule, not a guideline.
- **Promotion is the mechanism for turning individual insights into team intelligence.** A skill that lives only in a developer's User Layer benefits only that developer. A skill that has been reviewed and merged into the shared Skill Layer benefits the entire program, on every pull, forever.

> **Note:** The directory structure and file templates in this section are starting points, not mandates. Programs with different SAP landscapes, different numbers of business areas, or different governance structures should adapt the layer boundaries to fit their context. The essential properties — no credentials in the repository, a clear distinction between shared and personal configuration, and a pull-based distribution mechanism — should be preserved regardless of how the directory layout evolves.

Context - the knowledge graph

Imagine you have just joined a large enterprise engagement. The client has spent years building their technology landscape — custom naming conventions, proprietary domain terminology, carefully designed data models, architectural decisions encoded in dozens of documents scattered across SharePoint, Confluence, and email threads.

Now you open your agentic IDE and type: `"Generate a data access layer for the Customer Order domain."`

The agent responds — quickly, fluently, confidently. It produces code. It looks reasonable. It even looks professional.

But something is wrong.

The entity is called `CustomerOrder` — but your client's standard is `CustOrdTxn`. The repository pattern used doesn't match the approved framework. The error handling references a generic exception class, not the client's internal `EnterpriseException` hierarchy.

The field names don't align with the canonical data dictionary that the architecture team spent six months ratifying.

The agent didn't fail. It just didn't *know* what exactly is expected.

It generated from its general training — not from your project reality. And now someone has to review every line, reconcile every deviation, and explain to the client why the AI-generated code needs to be reworked before it can be used.

This is the context gap — the distance between what the agent knows generally and what your project requires specifically. And in enterprise engagements, that gap is wide.

IBM Bob is capable. But capability without context is limited. An AI assistant that can generate code but does not know your organization's architecture, your data model, or your business rules will produce technically correct but organizationally inappropriate answers.

The root cause: Agents without organizational memory

The agent is not broken. It is doing exactly what it was designed to do — generate plausible, well-structured output based on patterns it has learned. The problem is that *your organization's knowledge lives outside the agent's awareness*.

That knowledge exists in many forms:

- **Domain schemas**: the entities, relationships, and terminology that define your business
- **Project knowledge graphs**: the interconnected facts specific to this client, this engagement, this codebase
- **Architectural decisions**: the approved patterns, banned technologies, and design rationale
- **Data dictionaries**: the canonical names, types, and descriptions of every field that matters

Without a mechanism to bring this knowledge *into* the agent's reasoning context, every prompt starts from zero. Every generation is a guess. And every guess has to be reviewed, corrected, and re-aligned — erasing much of the productivity the agentic IDE was supposed to deliver.

What practitioners need is not just a smarter agent. They need a way to make the agent *organizationally aware* — grounded in the specific, structured knowledge that governs their project.

Context Studio is IBM Bob's answer to the context gap.

Described simply as a tool to *"manage domain schemas and client project knowledge graphs"*, Context Studio is in practice far more than a data management utility. It is the **organizational memory layer** for the agentic IDE — the mechanism by which enterprise knowledge is captured, structured, and made directly consumable by IBM Bob at runtime.

Context Studio is organized around two tabs that represent two progressive stages of knowledge operationalization: **Schema** and **Context**.

Stage 1 — Schema: Giving shape to what you know

Before the agent can use organizational knowledge, that knowledge must be structured. The **Schema** tab is where this happens.

A schema in Context Studio is not a database schema in the traditional sense. It is a formalized representation of *your organization's domain knowledge* — the entities your business cares about, the relationships between them, the terminology your teams use, and the patterns that define how your systems are built.

Context Studio provides four capabilities to build this schema:

Bring your organization knowledge and build a schema — Practitioners can directly author and encode the domain knowledge that defines their engagement. This might be the customer's canonical data model, the approved microservice taxonomy, or the glossary of terms that distinguishes their business from any other.

Analyze patterns, trends, and insights from documents — Rather than building from scratch, practitioners can feed existing documents — architecture decision records, data dictionaries, API specifications, design documents — into Context Studio. It extracts entities, relationships, and insights automatically, turning unstructured knowledge into structured schema elements.

Manage knowledge assets — All schema elements are governed centrally, giving teams a single source of truth for the domain knowledge that will ground the agent's behavior.

Review extracted entities and relationships — Automated extraction is a starting point, not an endpoint. This capability allows practitioners to validate, curate, and refine what was extracted — ensuring accuracy before it influences agent output.

The Schema tab answers the foundational question: *What does our organization know, and how is it structured?*

Stage 2 — Context: Activating knowledge for the agent

A schema on its own is inert. Its value is only realized when it is activated — transformed into a live, queryable context that the agent can draw upon during task execution. This is the purpose of the **Context** tab.

Build a comprehensive context for your projects — Project context is assembled from the schema, scoped to the specific engagement, client, or workstream at hand. This ensures the agent is not reasoning over generic domain knowledge but over *your project's* specific reality.

Use the published schema to create context — Context is always anchored to a published schema, maintaining traceability and consistency between the domain definition and the project-level context.

Create knowledge graphs by ingesting additional data — Context grows richer over time. Additional data sources — sprint artifacts, newly discovered architectural constraints, updated data models — can be ingested to continuously deepen the knowledge graph.

Expose the context as MCP tools and use it in applications — This is the integration point that connects Context Studio to IBM Bob. The knowledge graph is published as **MCP (Model Context Protocol) tools** — structured, callable interfaces that AI agents can invoke at runtime to query organizational knowledge directly.

The Context tab answers the operational question: *How does our structured knowledge become accessible to the agent in real time?*

The integration: Closing the gap

The Knowledge Graph integration addresses this gap by giving IBM Bob access to a persistent, structured representation of organizational knowledge. Rather than re-explaining context in every conversation, the relevant knowledge is already available — surfaced automatically when Bob needs it, kept current as the organization's systems and processes evolve.

This knowledge can encompass system architecture, data model documentation, business process definitions, integration maps, development standards, and decision records. It is not a static document — it is a living graph that grows more useful as it is used and maintained.

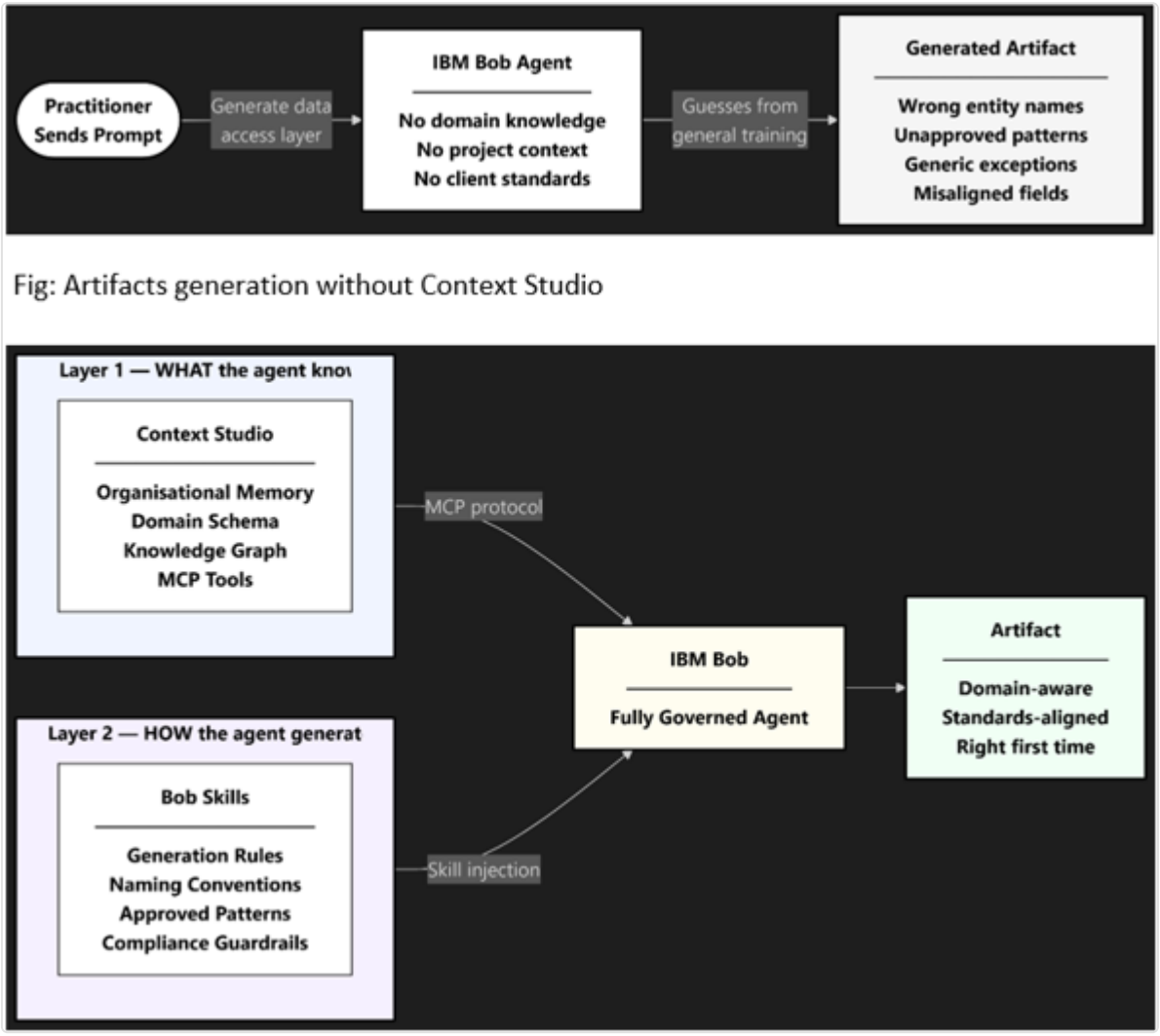


Figure: Artifacts Context Studio

When IBM Bob is configured with Context Studio's MCP tools, the agent no longer starts from zero. Every task is grounded in the following knowledge:

- The **canonical entities and relationships** defined in the schema
- The **project-specific knowledge graph** built for this engagement
- The **domain terminology** your client uses
- The **architectural facts** that constrain what should and should not be generated

Combined with **Bob Skills** — which govern *how* the agent generates — Context Studio governs *what the agent knows*. Together, they eliminate both dimensions of the context gap:

Dimension	Solved By
Agent is unaware of domain	Context Studio — injects structured organizational knowledge via MCP
Agent doesn't follow standards	Bob Skills — ingests project guidelines and generation rules into agent context

From gap to governed intelligence

Returning to our opening scenario — the practitioner who asked for a data access layer and received something that looked right but wasn't — the outcome is now entirely different.

Connecting Context Studio to IBM Bob produces the following results:

- The agent queries the knowledge graph and retrieves `CustOrdTxn` as the correct entity name
- It finds the approved repository pattern documented in the architectural schema
- It references the Enterprise Exception hierarchy recorded as a known knowledge asset
- It generates field names that match the canonical data dictionary exactly

And with a Bob Skill active, the naming convention, file structure, and code organization all conform to the project standard — without the practitioner having to specify any of it in the prompt.

The gap is closed. The agent is no longer guessing. It knows what your organization knows — and it generates accordingly.

This is the promise of Context Studio integrated with IBM Bob: not just a faster agent, but a contextually intelligent one — one that earns the practitioner's trust by generating artifacts that are right the first time.

Key takeaways

• Design a schema to capture entities and relationships within your application

• Build and publish context as an MCP endpoint for AI tooling to consume

• Integrate with IBM Bob to get instant, accurate application insight before coding

• Replace manual codebase scanning with context-driven understanding during implementation and enhancement work

IBM Context Studio provides a structured approach to understanding your application landscape before writing a single line of code. By designing a schema that captures the key entities and relationships within your system, you can build a rich context model that reflects the true shape of your application. Once assembled, this context is exposed as an MCP (Model Context Protocol) endpoint, making it available to AI-powered tooling in a standardized, consumable format. Integrating this context with IBM Bob transforms the way you approach implementation and enhancement work. Rather than manually scanning through codebases and piecing together an understanding of how components relate, IBM Bob draws on the pre-built context to give you immediate, accurate insight into your application's structure, dependencies, and business intent. Whether you are introducing new functionality or applying a targeted fix, this approach ensures your changes are grounded in a deep, reliable understanding of the system — reducing discovery time and improving the precision of every contribution.

For more information, see the IBM Context Studio section of the Additional Resources.

Spec driven development

Spec-Driven Development (SDD) is a structured methodology that positions specifications as executable, version-controlled blueprints for AI-assisted engineering. Rather than jumping directly from informal requirements to code generation, SDD establishes formal contracts between business intent, technical architecture, and implementation—ensuring that IBM Bob builds exactly what is specified with full traceability and alignment.

The Problem

Why Traditional ABAP Development Fails It's Tuesday morning. Thomas opens a JIRA ticket: '`ZSD_SALES_ORDER_ENHANCEMENT` — Add customer-specific pricing logic for new EU compliance rules.' The description is three sentences. The functional spec is a PDF from 2019, last updated by a consultant who left two years ago. The code he must modify — `ZCL_SO_PRICING_ENGINE` — is 2,847 lines of nested IF statements, Hungarian notation that changed convention halfway through, and comments that lie. "Just add a new condition type," the ticket says. Thomas

knows it's never that simple. He spends two hours reading code. Another hour searching for the right BAdI. After lunch, he codes — but halfway through, he realizes the pricing logic also touches credit check, which touches ATP check, which touches delivery creation. A change here ripples across five modules he didn't know existed. He codes defensively. Adds more IF statements. More exception handling. More comments that will lie to the next developer. By Friday, unit tests pass. But Thomas doesn't feel good. He knows what's coming: integration test in QA reveals a side effect in invoice generation. Business asks why the requirement wasn't clearer. The functional consultant says 'it was in the spec' — pointing to a paragraph that says 'handle EU pricing appropriately.' This is the problem. Not Thomas's skill. Not the tools. The problem is the **gap between what the business wants and what the developer builds** — a gap that widens with every handoff, every assumption, every 'it's obvious' that wasn't.

Visualizing the problem: Traditional SAP development flow

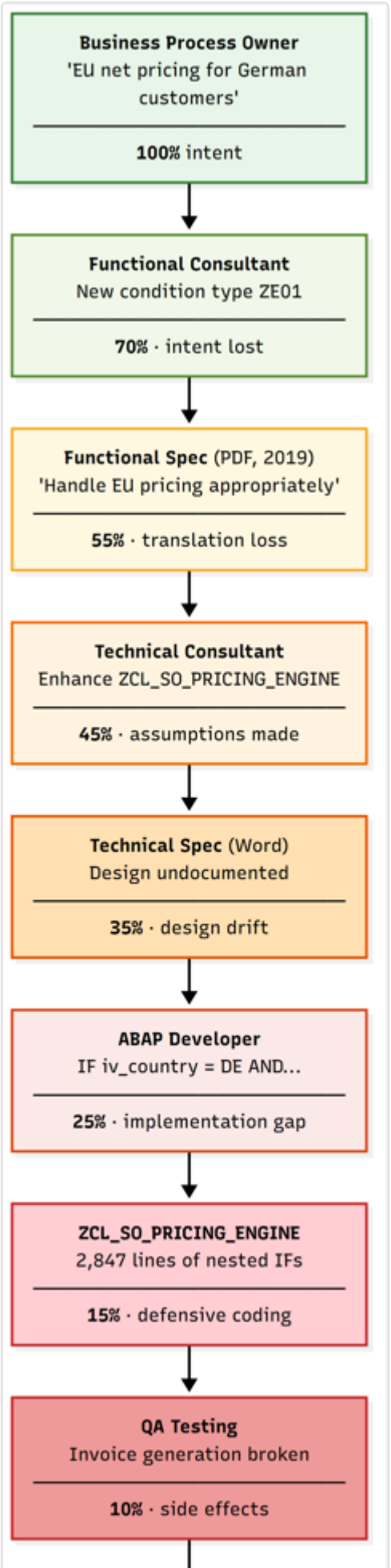

Business Process Owner
'EU net pricing for German customers'
100% intent
Functional Consultant
New condition type ZE01
70% · intent lost
Functional Spec (PDF, 2019)
'Handle EU pricing appropriately'
55% · translation loss
Technical Consultant
Enhance ZCL_SO_PRICING_ENGINE
45% · assumptions made
Technical Spec (Word)
Design undocumented
35% · design drift
ABAP Developer
IF iv_country = DE AND...
25% · implementation gap
ZCL_SO_PRICING_ENGINE
2,847 lines of nested IFs
15% · defensive coding
QA Testing
Invoice generation broken
10% · side effects

Figure: Traditional SAP Development Flow — Intent Lost at Every Handoff

Key Insight: At each handoff (arrow), 20-30% of original intent is lost. By the time code is written, the business need has been translated four times — each translation introducing assumptions, ambiguity, and drift.

The specification gap — Where requirements go to die

In a typical SAP project, the journey from business need to production code looks like this:

Business Owner → Functional Consultant → Functional Spec (PDF) → Technical Consultant → Technical Spec (Word) → ABAP Developer → Code → Transport → QA → Production

At each arrow, information is lost. The business owner speaks in outcomes: 'Customers in Germany need net pricing with tax breakdown.' The functional consultant translates to process: 'New condition type ZE01.' The technical consultant translates to objects: 'Enhance ZCL_SO_PRICING_ENGINE.' The developer translates to syntax: 'IF iv_country = DE...' By the time it reaches code, the original intent is buried under layers of interpretation.

The four gaps

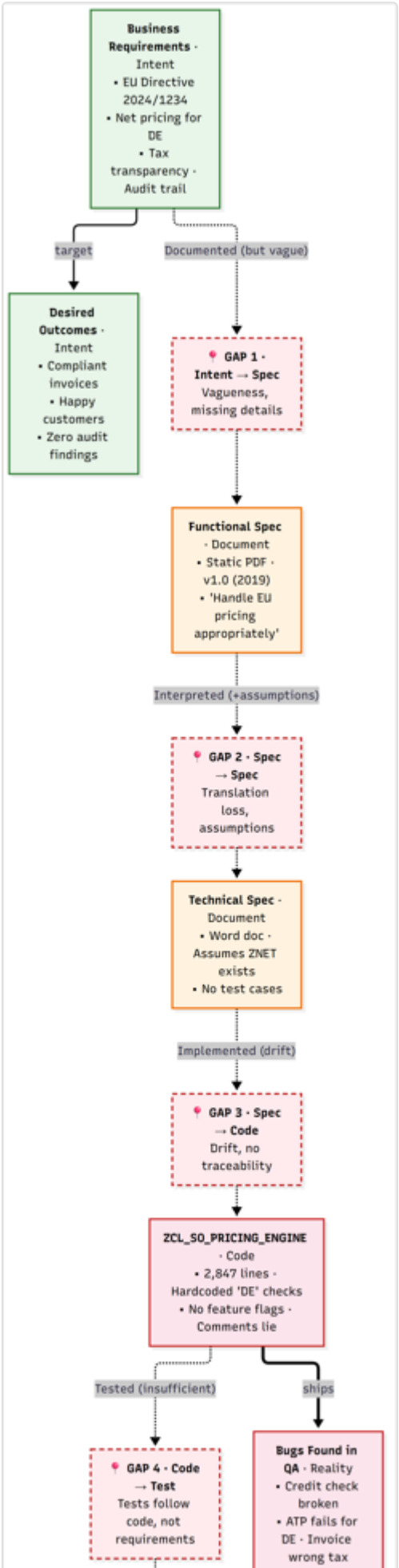

Business Requirements · Intent
• EU Directive 2024/1234
• Net pricing for DE
• Tax transparency · Audit trail
target
Documented (but vague)
Desired Outcomes · Intent
• Compliant invoices
• Happy customers
• Zero audit findings
GAP 1 · Intent → Spec
Vagueness, missing details
Functional Spec · Document
• Static PDF · v1.0 (2019)
• 'Handle EU pricing appropriately'
Interpreted (+assumptions)
GAP 2 · Spec → Spec
Translation loss, assumptions
Technical Spec · Document
• Word doc · Assumes ZNET exists
• No test cases
Implemented (drift)
GAP 3 · Spec → Code
Drift, no traceability
ZCL_SO_PRICING_ENGINE · Code
• 2,847 lines · Hardcoded 'DE' checks
• No feature flags · Comments lie
Tested (insufficient)
ships
GAP 4 · Code → Test
Tests follow code, not requirements
Bugs Found in QA · Reality
• Credit check broken
• ATP fails for DE · Invoice wrong tax

Figure: The Specification Gap — Four Distinct Gaps Where Intent Evaporates

Traditional tools don't bridge these gaps — they widen them by keeping specs and code in separate worlds. The spec is a PDF in SharePoint; the code is in the SAP system. They never sync.

The Solution — Spec-Kit Mode The answer isn't to write better PDFs. The answer is to make the specification the living center of your development workflow — versioned, validated, executable, and traceable. IBM Bob's **Spec-Driven Development with Spec-Kit Mode** does exactly this. Spec-Kit Mode closes all four gaps simultaneously: it creates structured specifications (Gap 1), validates them against your SAP domain skills (Gap 2), generates code that traces directly back to every requirement (Gap 3), and produces tests that map 1-to-1 to your acceptance criteria (Gap 4).

What is Spec-Kit mode?

Spec-Kit is a globally installable custom mode in IBM Bob — not a built-in. You install it once from Bob Settings → Modes, and it then appears in the chat mode dropdown alongside Agent, Plan, and Ask. The mode works together with the spec-kit CLI tool (`specify-cli`) which must also be installed and initialized per project to enable the `/speckit.*` slash commands. Think of it as Bob's SDD autopilot — once the mode is active and the CLI is initialized in your project, it orchestrates the entire spec-first lifecycle across 8 phases: constitution → specify → clarify → plan → checklist → tasks → analyze → implement, with built-in approval gates that prevent skipping steps.

How Spec-Kit differs from Plan / Agent / Ask

Capability	Plan Mode	Agent Mode	Spec-Kit Mode
Primary Role	Architect / Designer	Implementer / Coder	SDD Orchestrator (end-to-end)
Workflow Enforcement	Advisory — you decide	Advisory — you drive	Mandatory — enforces sequence
Specification Handling	Reads spec you provide	Reads spec + plan	Manages full spec lifecycle
Skill Integration	Loads on request	Loads on request	Auto-loads from spec context

Figure: How Spec-Kit Differs from Plan, Agent, or Ask

Finding and activating Spec-Kit mode

Complete all three steps below before using any `/speckit.*` command in Bob chat.

Spec-Kit Setup with IBM Bob

Table: Spec-Kit setup steps with commands and verification

Step	What	How / Command	Verify / Frequency
1 — Install Bob Mode	Adds Bob's 8-phase SDD workflow, approval gates, and phase sequencing.	Bob VS Code → Settings → Modes → Search Spec-Driven Development with Spec-Kit → Install	√ Mode appears in chat dropdown. Once — global to all workspaces.
2 — Install spec-kit CLI	Installs the specify CLI, SDD methodology, templates, validation rules, and slash commands.	uv tool install specify-cli --from git+https://github.com/github/spec-kit.git	√ specify version displays the version. Once — per machine.
3 — Initialize Project	Creates .specify/ and installs /speckit.* commands into .bob/commands/. Enables the SDD workflow for the project.	echo "y"	specify init . --ai bob

Table: Spec-Kit capabilities enabled by installation

Capability	Description
Spec-First Workflow	Prevents jumping directly to implementation without a validated specification.
SDD Slash Commands	Provides the complete 8-command SDD lifecycle from /speckit.constitution through /speckit.implement.
Domain Skill Discovery	Reads specification context and loads relevant domain skills such as S4SalesSkill, S4ProcurementSkill, and S4FinanceSkill.
Traceability	Links requirements and acceptance criteria to artifacts and tests.
Continuous Validation	Supports spec linting, plan completeness, code-to-spec conformance, and test-coverage validation.

No restart, no configuration. The prompt hint changes to reflect SDD workflow commands.

Now that Spec-Kit CLI is installed switch Bob chat to **Spec-Driven Development with Spec-Kit** mode → type `/speckit.constitution` to begin.

Part 4: Spec-Kit Activation Flow and Commands

1	Open IBM Bob in VS Code	Click the mode selector dropdown in the Bob chat panel — shows 'Agent' by default.
2	Select Spec-Driven Development with Spec-Kit	Choose 'Spec-Driven Development with Spec-Kit' from the dropdown. No restart, no configuration needed.
3	Bob Loads Spec-Kit Skill	Automatically loads SDD methodology, specification templates, validation rules, and all phase commands.
4	Auto-Detects Domain from Workspace	Scans Git repo structure, existing specs, and skill manifests to identify the business domain.
☑	Spec-Kit Mode Active	Prompt hint changes to Spec-Kit commands. Ready: type /spec:create to begin or /spec:validate to check an existing spec.

Figure: Spec-Kit Mode Activation Flow

Spec-Kit commands at a glance

Command	Phase	Purpose
/spec:create	1. Specify	Interactive wizard — generates SPEC.md with Business Context, FRs, NFRs, Acceptance Criteria
/spec:validate	2. Validate	Linting + traceability check + skill alignment verification
/spec:plan	3. Plan	Impact analysis + implementation sequence + risks + traceability matrix
/spec:implement	4. Implement	ABAP code generation + test classes + inline FR/AC traceability references
/spec:test	5. Test	Execute ABAP Unit tests, report coverage against each Acceptance Criterion
/spec:deploy	6. Deploy	Create transport request with spec version tag and feature flag documentation

Figure: Spec-Kit Commands at a Glance

Spec-Kit end-to-end workflow

This is the complete SDD lifecycle enforced by Spec-Kit Mode. Each phase must complete before the next begins — guardrails prevent skipping.

Phase	Command	Input	Output	Guardrail
1. Specify	**/spec:create**	Business Requirement	SPEC.md (validated)	No empty sections
2. Validate	**/spec:validate**	SPEC.md	Validated Spec	100% traceability FR↔AC
3. Plan	**/spec:plan**	Spec + Codebase + Skills	PLAN.md + Matrix	All objects identified
4. Implement	**/spec:implement**	Plan + Spec + Skills	ABAP Code + Tests	Clean Code + traceability
5. Test	**/spec:test**	Code + Tests	Results + Coverage	100% AC coverage
6. Deploy	**/spec:deploy**	Passing Tests	Transport + Docs	Zero ATC findings

Figure: Spec-Kit End-to-End Workflow

Phase-by-phase walkthrough — EU pricing example

The following example shows how enabling Spec-Kit mode allows for the use of specific specs in generating a solution rather than relying on an individual user prompt.

Phase 0: Prerequisites and Project Initialization Before running any /speckit.* command in Bob chat, you must run ONE terminal command to initialize the project. This wires spec-kit into your workspace and installs all slash commands.

Command to run (in terminal, inside your project folder)	What it creates
echo "y" | specify init . --ai bob	Creates two folders in your project: .bob\commands\ (slash commands for Bob chat) and .specify\ (SDD artifacts folder).
After init — verify with:	specify check — prints 'Specify CLI is ready to use!' if everything is wired correctly.
Folder: .bob\commands	Contains all /speckit.* slash command files that Bob reads from. You will see: speckit.constitution.md, speckit.specify.md, speckit.clarify.md, speckit.plan.md, speckit.checklist.md, speckit.tasks.md, speckit.analyze.md, speckit.implement.md, speckit.converge.md, speckit.taskstoissues.md
Folder: .specify	Contains: memory\ (constitution.md lives here after Phase 1), templates\ (spec/plan/task templates), scripts\ (workflow automation), workflows\ (phase definitions), integrations\ (tool integrations), init-options.json, integration.json
Now switch Bob to Spec-Kit mode	Click the mode dropdown in Bob chat → select 'Spec-Driven Development with Spec-Kit' → type /speckit.constitution in the chat box to begin Phase 1.

Figure: Prerequisites — Project Initialization

Phase 1: Define Project Principles — `/speckit.constitution`

What /speckit.constitution asks	EU Pricing Example
Architecture rules (What tech stack must ALWAYS be used)	RAP managed scenario, OData V4 only, Fiori Elements, released SAP APIs only — no classic BAPIs
Coding standards (What naming and quality rules apply)	Object prefix ZSD_, max 250 lines per class, no SELECT *, no hardcoded values, feature flags for all new logic
Testing policy (What test standards are mandatory)	ABAP Unit Tests mandatory for all validations and actions, minimum 80% statement coverage, test doubles for all SAP API dependencies
Security rules (Who sees what)	Pricing amounts visible only to roles: ZSD_PRICING_MANAGER. Sales reps see APPROVED/BLOCKED status only — not actual figures
Transport & deployment (What governance applies) **.specify\scripts** **.specify\workflows** **.specify\integrations** **.specify\init-options.json** **.specify\integration.json**	All objects in transport request before deployment. No manual activation in production. All changes via CTS transport only

Figure: Define Project Principles

Output: Bob saves your answers to `.specify/memory/constitution.md` — loaded automatically at every subsequent phase. You never need to repeat these rules.

> **Attention:** Do NOT skip this phase. Without a constitution, Bob generates ABAP without your organization's naming prefix, testing rules, or architecture constraints.

Phase 2: Define the Specification — `/speckit.specify`

Spec Section	Content	EU Pricing Example
Business Context	**Regulation, effective date, scope**	EU Directive 2024/1234 · Effective 2025-01-01 · DE B2C sales orders
FR-01: Net Price Display	**WHEN / AND / THEN statement**	WHEN German customer + NET pricing → show net, tax, gross separately
FR-02: Condition Type ZE01	**New object definition**	Access sequence: Customer → Material → Generic · Net weight basis
FR-03: Integration	**Touch points with existing objects**	Credit check (ZCL_SO_CREDIT_CHECK) · ATP · Output type ZEUINV
Acceptance Criteria	**Testable pass/fail conditions**	AC-01: DE net price shown · AC-02: FR unchanged · AC-03: Country change recalc · AC-04: ATC clean

Figure: Define Specification

Output: `specs/001-eu-pricing/spec.md` — versioned in Git, reviewable in pull request.

Phase 3: Validate the Specification — `/speckit.clarify` (Optional but Strongly Recommended)

Run `/speckit.clarify` in Bob chat. Bob reads `spec.md` and runs 5 automated checks — this is the quality gate between specification and technical design. Example output for EU Pricing:

Check	Result · Detail
Completeness	☑ PASS — All sections present: Business Context, FRs, NFRs, Acceptance Criteria
Consistency	☑ PASS — FR-02 references ZE01; AC-01 tests ZE01; no orphan requirements
Traceability	☑ PASS — Each FR maps to ≥1 AC; each AC maps to testable, measurable behaviour
Skill Alignment	☑ PASS — 'Access sequence' matches S4SalesSkill; V_VBAK_VKO auth object exists
⚠ Warning	⚠ FLAG — FR-03 mentions output type ZEUINV but no FR defines output determination logic → Bob suggests adding FR-04
Action Required	Add FR-04 (Output Determination) to spec.md → re-run /speckit.clarify → all checks pass → proceed to /speckit.plan. This warning prevents a gap that would have caused rework in QA.
Result	Spec is now implementation-ready. Proceed to Phase 4: /speckit.plan

Figure: Validate the Specification

Phase 4: Generate Technical Plan — `/speckit.plan`

Bob reads the validated spec, queries your SAP system via ADT MCP, and loads `S4SalesSkill`. Produces a surgical implementation plan with impact analysis, implementation sequence, risk mitigation, skill dependencies, and traceability matrix. Output: `specs/001-eu-pricing/plan.md`

Phase 5: Generate Task Breakdown — `/speckit.tasks`

Bob converts `plan.md` into an ordered, atomic, checkable task list saved to `specs/001-eu-pricing/tasks.md`. REQUIRED before `/speckit.implement` — Bob executes from `tasks.md`, not `plan.md`.

```
CLASS zcl_so_pricing_engine DEFINITION PUBLIC FINAL CREATE PUBLIC.
  PUBLIC SECTION.
    INTERFACES: zif_so_pricing_engine.
    METHODS: determine_german_net_price
      IMPORTING iv_vbeln TYPE vbeln_vl iv_posnr TYPE posnr_vl
      RETURNING VALUE(rs_result) TYPE zsd_s_german_net_price
      RAISING zcx_pricing_error.
  PRIVATE SECTION.
    METHODS: get_condition_ze01 ... calculate_german_vat ...
    DATA: mv_active TYPE abap_bool VALUE abap_false.  " Feature flag
ENDCLASS.

" FR-01: Net Price Display  |  FR-02: Condition Type ZE01  |  AC-01: DE net price
```

Figure: Bob Generates ABAP

Phase 6: Implement — `/speckit.implement`

Bob executes tasks from `tasks.md` in batches of 5, pausing after each batch for your approval. Generates complete, production-ready ABAP with inline spec traceability (FR-01, AC-01 comments on every method).

Test Method	Maps To	Result
test_german_net_price	AC-01	☑ PASS
test_french_unchanged	AC-02	☑ PASS
test_country_change_recalc	AC-03	☑ PASS
test_atc_compliance	AC-04	☑ PASS
Coverage Report	—	**94% statements · 87% branches · 100% AC coverage**

Figure: Implement — /speckit.implement

Phase 7: Spec-Kit in Action — Procurement Tolerance Example

Scenario: "Add plant-specific tolerance limits for invoice verification (MIRO) with automatic workflow routing."

Step	Action
Thomas:	/speckit.specify
Bob:	"Domain?" → Thomas: "Procurement / Invoice Verification" → Loads S4ProcurementSkill
Bob:	"Trigger?" → Thomas: "Plant 1000 ±2%, Plant 2000 ±5%, else workflow"
Bob:	Generates SPEC_ZMM_INV_TOLERANCE.md: FR-01 config table, FR-02 MIRO check, FR-03 workflow, AC-01..04
Thomas:	/speckit.clarify → Clean
Thomas:	/speckit.plan → Reads ZMM_INV_VERIFY_ENH, queries MARA/WERKS_D, 5-step plan
Thomas:	/speckit.implement → Generates code below

Figure: Spec-Kit in Action — Procurement Tolerance Example

```
CLASS zcl_mm_inv_tolerance_check DEFINITION PUBLIC FINAL CREATE PUBLIC.
  PUBLIC SECTION.
    INTERFACES: zif_mm_inv_tolerance_check.
    METHODS: check_tolerance
      IMPORTING iv_werks TYPE werks_d iv_invoice_amt TYPE wrbtr iv_po_amt TYPE wrbtr
      RETURNING VALUE(rs_result) TYPE zmm_s_tolerance_result
      RAISING zcx_mm_inv_error.
  PRIVATE SECTION.
    METHODS: get_tolerance_config ... route_to_workflow ...
ENDCLASS.

" FR-01: Plant config table  |  FR-02: MIRO tolerance  |  FR-03: Workflow routing
" Auth: M_MSEG_WERKS  |  API: SAP_WAPI_START_WORKFLOW  |  Config: ZMM_TOLERANCE_CONFIG
```

Figure: Code Snippet

The following items were delivered without Thomas writing a single code prompt:

✅ Clean Code: `FINAL`, `CREATE PUBLIC`, interface segregation

✅ Hungarian: `iv_`, `ls_`, `lv_`, `rs_`

✅ Domain auth: `M_MSEG_WERKS` (from `S4ProcurementSkill`)

✅ Config table pattern: `ZMM_TOLERANCE_CONFIG` with fallback defaults

✅ Standard SAP workflow: `SAP_WAPI_START_WORKFLOW`

✅ Inline traceability: Every method maps to FR-01/02/03

✅ Exception class: `ZCX_MM_INV_ERROR`

Then `/speckit.implement` generates test class mapping to AC-01..04. All pass. ATC clean. Transport created with spec version tag. Done.

Phase 8: Guardrails and Traceability — What Makes Spec-Kit Different

Guardrail	Enforcement	Benefit
No implement without valid spec	/speckit.implement blocks if /speckit.clarify not passed	Prevents code-spec drift
No deploy without passing tests	[SAP Transport — handled outside spec-kit via standard CTS process] blocks if /speckit.implement (testing is part of the implement phase) has failures	Quality gate enforced
No phase skipping	Commands must run in sequence: create -> validate -> plan -> implement -> test -> deploy	Discipline by default
Auto-traceability matrix	Every artifact links to FR/AC; maintained automatically	Audit-ready, zero manual effort
Spec version in transport	spec-version tag embedded in transport request	Full lifecycle traceability

Figure: Guardrails and Traceability — What Makes Spec-Kit Different

Phase 9: Before vs After — Thomas's Transformation

Three months later. Same company. Same Thomas. New ticket: "ZSD_EU_PRICING — Extend to Austria (AT) with 20% VAT."

❌ BEFORE: Traditional Approach	☑ AFTER: Spec-Driven Development with Spec-Kit
1. Tuesday: JIRA ticket 'Just add condition type' 2. 2 hours: Read 2,847 lines of legacy spaghetti code 3. 1 hour: Search for BAdI — which enhancement spot? 4. Lunch 5. Code... discover ripple: Credit check → ATP → Delivery (5 modules!) 6. Defensive coding: More IFs, more exceptions, more lying comments 7. Friday: Unit tests pass but Thomas uneasy... 8. QA: Integration fails — invoice generation broken 9. Business: 'Not what we asked!' Consultant: 'Was in spec!' 10. Rework: 2-3 sprints, frustration all around	1. Tuesday: Update SPEC.md — add Austria to FR-01, FR-02, add FR-04, AC-05 2. PR: Functional consultant 'Yes, exactly what we need' | Tech lead 'Plan looks solid' 3. /speckit.clarify → Clean 4. /speckit.plan → Bob reads diff + code → surgical plan 5. /speckit.implement → Bob implements in minutes 6. /speckit.implement (testing is part of the implement phase) → Tests pass, ATC clean, Transport created 7. Lunch: Change in QA 8. Afternoon: Business validates AC-05 9. End of day: Approved for production 10. Thomas goes home on time. Spec updated. Code matches. 11. Next month: Junior hire opens spec → understands intent → modifies confidently

Figure: Before vs After — Thomas's Transformation

Phase 10: When to Use Which Mode

Scenario	Recommended Mode	Why
Greenfield / New enhancement	**Spec-Kit Mode**	Full lifecycle, guardrails, zero prompt engineering
Regulation-driven / Audit-critical	**Spec-Kit Mode**	Enforced traceability, living spec, audit trail
Team handoff / Knowledge transfer	**Spec-Kit Mode**	Spec = documentation; next dev reads intent in hours
Exploring / Spike / Prototype	**Agent Mode**	Flexible, conversational, no spec overhead
Architecture review / Design discussion	**Plan Mode**	Collaborative, iterative, human-in-the-loop

Figure: When to Use Which Mode

Phase 11: Key Takeaways for ABAP Teams

Discipline	You can't skip spec validation. You can't deploy without tests.
Speed	What took 3 days (spec → plan → code → test → transport) now takes hours.
Consistency	Every enhancement follows the same pattern. Same spec structure. Tests map 1:1 to AC.
Traceability	Built-in, automatic, auditable. No manual matrix maintenance.
Knowledge Capture	The spec IS the documentation. Next developer reads spec → understands intent → modifies confidently.

Figure: Key Takeaways for ABAP Teams

> TIP: Not sure where to begin your Spec-Kit journey, or unsure how to run any of the phase commands? Simply ask **IBM Bob** — to walk you through each step with clear explanations and guided learning tailored to your context. IBM Bob can demonstrate how each `/spec:*` command works, clarify what inputs are required, and help you understand what to expect at every stage of the lifecycle.

For more information, see the Spec-Kit (Spec-Driven Development) section of the Additional Resources.

Development with IBM Bob

SAP development is not a single activity. It spans ABAP program authoring, CDS view design, OData service scaffolding, UI5 Fiori application construction, SAP Integration Suite flow development, and the documentation that ties all of these together into a maintainable, auditable project record. IBM Bob operates across that full span — not as a generic code completion tool, but as a configurable, codebase-aware development partner that applies project-specific standards to every output it produces.

Two capabilities distinguish Bob from tools that generate code in isolation. First, Bob reads the actual project codebase before generating anything. It fetches the live ABAP source from the connected SAP system, inspects the existing OData service definition, or reads the CPI integration flow — and generates code that fits the reality of what is already there, not a textbook approximation of what might be there. Second, Bob enforces what the project has decided. Workspace rules, custom skills, and project-specific modes are loaded into every conversation automatically, which means a developer three months into a project and a consultant joining on day one both receive output that conforms to the same standards, without either of them having to know the full rule set by heart.

The governance model that makes this safe in practice is the same one a senior architect would design independently: read access to live systems is granted by default; write and activation operations require explicit approval and are gated by hooks that enforce package rules, documentation headers, and syntax constraints before anything reaches the transport layer. AI assistance and change control are not in tension here. They are designed to work together.

What this section covers

This section is organized around the development activities an SAP project encounters in the order it encounters them — from the first prompt a developer submits through to the published project documentation that the implementation leaves behind.

ABAP development opens the technical content, covering prompt engineering discipline, the two VS Code extensions that connect Bob to SAP (ABAP ADT and ABAP FS), and the full lifecycle from object creation through code generation, review, and activation. The treatment is practical: every pattern is grounded in a real ABAP artifact — an ALV purchase-order report, a 900-line OData DPC Extension class, a BAdI implementation for pricing logic — rather than a synthetic example constructed for illustration.

SAP BTP UI5 development follows, with the SAP Cloud Application Programming Model (CAP), Node.js, and SAP Business Application Studio (BAS) as the platform context. Because BAS is a cloud-hosted IDE rather than a local file system, connecting an external AI tool to it requires a deliberate integration design. Two architectures are documented — Outside-In, in which Bob runs locally and reaches into BAS through a custom MCP server, and Inside-Out, in which a Bob shell runs directly inside the BAS terminal. This section makes a direct recommendation about which to prefer for each class of project.

SAP Integration Suite development is the third technical area. A custom `ci-mcp-server` gives Bob read access to live CPI tenants, enabling flow inspection, Groovy script generation, failure diagnosis, and migration assessment from PI/PO landscapes. The architecture separates observation from mutation — Bob reads freely, but every proposed change is a draft until a developer explicitly approves it. That distinction is not cosmetic; it is the governance boundary that makes AI assistance responsible to extend to a live integration tenant.

SAP project documentation closes this section. Bob generates Functional Specifications, Technical Specifications, Business Process Procedures, data migration documents, and ABAP code review reports from source code, workshop notes, or WRICEF registers. The value is not speed alone. Every document generated by a correctly configured Bob instance follows the same structure, applies the same templates, and uses the same project-specific terminology — a consistency that is impossible to achieve at scale through manual authoring alone.

Code generation tips and tricks

Knowing how to write a prompt is only part of the picture. Getting consistently high-quality ABAP output from IBM Bob depends on a small set of repeatable habits — habits that shape the context Bob works in, the specificity of the instruction it receives, and the way work is broken down and reviewed.

This section documents those habits as concrete, actionable practices. Each one addresses a specific failure mode that teams encounter when they first start using AI-assisted code generation: prompts that are too vague to produce useful output, context windows that carry stale information from a previous task, missing project standards that force the same instructions to be repeated in every session, or complex objects attempted in a single request when they should be built incrementally.

The practices here apply primarily to SAP ABAP development, where the combination of a large legacy codebase, strict naming conventions, transport discipline, and SAP-specific APIs means that precision and context matter more than in greenfield development. Most of the patterns extend naturally to UI5, CAP, and CPI work, which are covered in their own sections later in this module.

Core practices

Be specific and structured in writing prompts

The more precise requirement, the better the generated code.

Table: Vague vs precise prompt examples

❌ Vague Prompt	✅ Precise Prompt
"Write an ALV report"	"Write a CL_GUI_ALV_GRID OO-based ALV report for EKKO/EKPO with a zebra layout, toolbar button, and double-click event handler"
"Fetch purchase order data"	"Select from EKKO/EKPO for company code P_BUKRS, date range P_DATFR–P_DATTO, guarded FOR ALL ENTRIES with IS NOT INITIAL check"
"Add error handling"	"Add AT SELECTION-SCREEN validation: from-date must not exceed to-date; all required parameters must be filled"

> **Tip:** Name the SAP table, the class/FM you want used, the pattern you expect, and the business rule — Bob will follow all of them exactly.

Use `@file` mentions to give Bob existing code as context

Instead of describing your existing program from memory, **reference it directly**:

```
@/ZPO_DETAILS_DISPLAY.prog.abap
Add a new selection-option S_LIFNR for vendor number and include it in the SELECT on EKKO
```

Bob reads the actual file, finds the right `SELECT` statement, and makes a **surgical targeted edit** — it will not rewrite unrelated code.

You can also scope to specific line ranges for large programs:

```
@/ZPO_DETAILS_DISPLAY.prog.abap:127-260
Optimize the FETCH_DATA form — replace individual READ TABLE calls with sorted table binary search
```

Use Plan mode first for complex ABAP objects

For large or architectural tasks, **start in Plan mode** (`/plan`) before switching to Agent:

```
/plan
Design a RAP Business Object for Purchase Requisition approval workflow:
- CDS root view, projection view
- Behavior definition with approve/reject actions
- Determination for status
- Validation for approval threshold
```

Bob produces a detailed implementation plan — review and confirm it, **then** switch to `/agent` to generate the code. This avoids wasted generation on the wrong approach.

Use Literate Coding mode inside your `.abap` file

Toggle **Literate Coding mode** (`Ctrl+I` on Windows) and write instructions **directly inside the ABAP source file** where the code should go:

```
FORM build_fieldcat.

  " Bob: build a field catalog for EKKO/EKPO ALV using a DEFINE macro.
  " Include EBELN and EBELP as key fields. Mark NETPR with currency reference WAERS.
  " Hide the technical ROW_COLOR field.

ENDFORM.
```

Press **Ctrl+Enter** → Bob replaces the comment with real code and shows an inline diff. Press **Ctrl+Enter** again to accept or **Ctrl+Shift+Backspace** to reject. No need to switch to the chat panel at all.

Enforce your SAP standards via `AGENTS.md`

Create an `AGENTS.md` file in your workspace root. Bob loads it automatically for **every conversation** — no need to repeat rules each time:

```
 ABAP Coding Standards

Database Access
- Always guard FOR ALL ENTRIES with IS NOT INITIAL check
- Never use SELECT * — list required fields explicitly
- Always include MANDT client field implicitly; do not add it to WHERE

 ALV
- Always use CL_GUI_ALV_GRID (never REUSE_ALV_GRID_DISPLAY)
- Field catalog built via DEFINE macro for compactness
- Layout: zebra = abap_true, cwidth_opt = abap_true, sel_mode = 'B'
- Register event handlers with SET HANDLER ... FOR <grid_ref>

Naming Conventions
- Programs: ZPR_<OBJECT>_<ACTION>
- Classes: ZCL_<OBJECT>_<PURPOSE>
- Function Groups: Z<AREA>_<TOPIC>
- Types: TY_<NAME>, internal tables: GT_/LT_, structures: GS_/LS_

 Messages
- Use MESSAGE TYPE 'S' DISPLAY LIKE 'E' inside ALV context — never TYPE 'E'
- No hardcoded message texts — use message class ZMM_MESSAGES

Performance
- Use SORTED or HASHED tables for lookup tables read in loops
- Avoid nested SELECT — use FOR ALL ENTRIES instead
```

Break large programs into focused incremental requests

Do **not** ask for an entire complex program in one shot if it has many independent pieces. Chain focused requests:

```
Step 1: "Generate type declarations and data declarations for the PO ALV report"
Step 2: "Now add the selection screen with blocks B1 and B2"
Step 3: "Add FETCH_DATA form — reads EKKO, EKPO, MSEG, MKPF, RBKP, RBSEG"
Step 4: "Add ALV display form with OO grid, field catalog, layout, event handler class"
Step 5: "Add DISPLAY_CHANGE_DOCUMENTS form with popup container"
```

Each step is reviewed and accepted before the next — giving you **full control** at every stage.

Ask Bob to refactor legacy procedural ABAP to OO

Bob is highly effective at modernization.

```
@/ZOLD_REPORT.prog.abap
Refactor this procedural ABAP report:
- Move all SELECT logic into a data provider class ZCL_PO_DATA_PROVIDER
- Move all ALV display logic into ZCL_PO_ALV_DISPLAY
- The main report program should only instantiate these two classes and call their methods
- Preserve all existing business logic exactly
```

Use `@problems` for syntax error fixes

After a failed activation in SE38, paste the error or use `@problems`:

```
@problems
Fix all syntax errors in the active ABAP file
```

Or paste the short dump / activation error directly:

```
Activation error in ZPO_DETAILS_DISPLAY:
  Line 329: Field "TXZ01" is not a component of structure GS_FIELDCAT
Fix this.
```

Bob locates the exact line and applies the minimal correction.

Generate test data / ABAP Unit test stubs

```
Generate ABAP Unit tests for FORM fetch_data in @/ZPO_DETAILS_DISPLAY.prog.abap:
- Test: empty selection returns empty GT_PO_DISPLAY
- Test: valid EKKO/EKPO returns one row with correct EBELN/EBELP
- Use CL_ABAP_UNIT_ASSERT
- Mock database calls using test doubles
```

Start a new task for unrelated changes

When you have finished one piece of work and move to a different program or topic, **always start a new Bob task** (+ New Task button). This prevents **context poisoning** — where stale code from a previous task causes Bob to make incorrect assumptions about your current file.

Each new task starts clean while keeping your `AGENTS.md` rules active.

Table: Prompt patterns for common ABAP tasks

Task	Prompt Pattern
BAdI implementation	"Create a BAdI implementation for BAdI definition . Filter by plant WERKS. Implement method to..."
BAPI call with error handling	"Call BAPI_PO_CREATE1 with header and item data from GS_PO. Handle RETURN table — collect E/A messages and display in ALV popup"
CDS View	"Create a CDS consumption view for EKKO/EKPO with association to LFA1 for vendor name. Expose for OData. Add value help for EBELN"
Smartform	"Generate a Smartform driver program for form ZORDER_CONF. Pass header WA_EKKO and items GT_EKPO as form interface"
Enhancement spot	"Add an implicit enhancement at the end of FORM MOVE_DATA in program SAPMV45A to populate a custom field ZZFIELD from a Z-table"

Quick impact summary

Table: Quick impact summary — tips, benefits, and time saved

Tip	Primary Benefit	Time Saved
Specific Prompts	Correct code on first generation	Eliminates 2–3 fix cycles
@file Mentions	Surgical edits, no collateral changes	Avoids full-file rewrites
Plan Mode	Right architecture from the start	Prevents design rework
Literate Coding	In-editor generation, no copy-paste	Faster iteration
AGENTS.md	Auto-enforced SAP standards	Zero per-session re-instruction
Incremental Steps	Full control at each stage	Easier review & acceptance
Refactoring	Legacy → OO ABAP without manual effort	Days → hours
@problems	Pinpoint syntax fixes	Instant activation fix
Unit Tests	Quality built in from day one	Fewer QA defects
New Task	Clean context, accurate generation	Avoids hallucination-based bugs
Prompt Patterns	Repeatable, team-wide consistency	Reusable across projects

SAP ABAP

The following sections discusses some tips and tricks related to SAP ABAP code generation showing recommended practices for prompt engineering in the ABAP code generation context.

Natural language to generate ABAP code

Bob can turn plain English descriptions into fully working ABAP programs. As demonstrated in this very session, you can provide a single prompt such as in the following example:

```
"Generate a report to display purchase orders with GR and invoices in an ALV report, with change document
```

The prompt produced a complete, structured ABAP program that included the following elements:

- Type declarations, selection screen, data fetching forms
- `cl_gui_alv_grid` with field catalog, layout, and toolbar
- Local class with event handler methods
- Change document popup with nested ALV
- GOS-based attachment using `cl_gos_manager`

Bob does not generate stub/placeholder code — it generates production-quality, syntactically correct ABAP.

Full ABAP lifecycle coverage

Bob supports the complete software development lifecycle, not just initial code generation:

Table: IBM Bob ABAP lifecycle capabilities

Lifecycle Phase	Bob Capability
Design	Plan mode — break down complex ABAP requirements into object model, table layout, program structure
Scaffolding	Generate complete program skeletons: REPORT, FUNCTION GROUP, CLASS, INTERFACE
Implementation	Write BAPIs, ALV reports, enhancements, user exits, BADIs, classes, FM
Refactoring	Clean up legacy procedural ABAP into OO ABAP classes
Debugging assistance	Explain runtime errors, dump analysis (short dumps), suggest fixes
Documentation	Generate inline comments, method documentation, technical specs from code
Review	Spot performance issues (missing indexes, SELECT *, no FOR ALL ENTRIES guard)

Deep ABAP language knowledge

Bob understands the full ABAP language stack:

Classic ABAP:

- `REPORT`, `FUNCTION-POOL`, selection screens, `PERFORM`/`FORM`, `INCLUDE` programs
- Database access: `SELECT`, `FOR ALL ENTRIES`, `JOIN`, `INTO TABLE`
- Internal tables: `STANDARD`, `SORTED`, `HASHED` with appropriate key usage
- ALV: `CL_GUI_ALV_GRID`, `REUSE_ALV_GRID_DISPLAY`, field catalogs, layouts, events

ABAP OO:

- Class definitions (`DEFINITION`, `IMPLEMENTATION`), interfaces, inheritance
- Exception classes, `TRY...CATCH`, `RAISE EXCEPTION TYPE`
- Design patterns in ABAP context (Factory, Observer, Strategy)

SAP-Specific Objects:

- BAPIs, RFCs, BDCs, IDocs
- Change documents (`CDHDR`/`CDPOS`), GOS/Attachments, workflow objects
- Enhancements: BAdIs, User Exits, Customer Exits, implicit/explicit enhancements
- SAPscript, Smart Forms, Adobe Forms
- CDS Views, AMDP, RAP (Restful ABAP Programming Model)

Literate coding — Write instructions inside the ABAP file

Bob supports **Literate Coding**: you can write natural language instructions directly inside your `.abap` source file, highlighted in blue, and Bob generates the actual code in-place with an inline diff for review.

Example in an ABAP context:

```
FORM fetch_vendor_data.
  " Bob: fetch all vendors from LFA1 for company code p_bukrs,
  " join with LFB1 for account group, sort by LIFNR
ENDFORM.
```

Bob replaces the comment with the actual `SELECT` statement and sorts.

Codebase-aware assistance

Bob reads your actual workspace files before answering and providing a response base on what is found. When working with an ABAP project it can provide the following functions:

- Read existing programs to understand naming conventions before generating new code
- Cross-reference type definitions across `TYPE-POOLS`, `INCLUDE` programs, and data dictionary
- Follow your existing coding style (indentation, comment format, variable naming)
- Detect conflicts with existing form names, class names, or global data

Surgical edits via *apply_diff*

Bob does not rewrite entire files unnecessarily. It uses targeted diff-based edits:

- Add a new `SELECT-OPTIONS` to an existing selection screen
- Insert a new column into an existing field catalog
- Fix a specific `READ TABLE` statement without touching surrounding code
- Rename a variable across a form block

Custom rules & ABAP project standards (*AGENTS.md*)

You can create an `AGENTS.md` file in your workspace to enforce SAP-specific coding standards for every Bob interaction:

```
# ABAP Project Rules
- Always use FOR ALL ENTRIES with IS NOT INITIAL guard
- Use CL_GUI_ALV_GRID (never REUSE_ALV_GRID_DISPLAY)
- All programs must have an AT SELECTION-SCREEN validation block
- Class names follow ZCL_<OBJECT>_<PURPOSE> convention
- No SELECT * — always specify required fields
- Use MESSAGE TYPE 'S' DISPLAY LIKE 'E' instead of TYPE 'E' in ALV
```

Bob will respect these rules on every code generation in the project.

Skills — Reusable ABAP instruction sets

Bob supports Skills — pre-packaged instruction sets that can be activated for specialist tasks. You could create an ABAP-specific skill that covers the following topics:

- ALV grid with OO event handler pattern
- RAP/CAP model generation standards
- Standard BAPI call wrappers with error handling
- CDS view generation conventions

Skills are loaded on demand and keep Bob's context lean.

Three modes for ABAP work

Table: IBM Bob modes for ABAP work

Mode	When to Use for ABAP
Plan	Design the program structure, table layout, class hierarchy, enhancement strategy before writing code
Agent	Write, modify, refactor ABAP — creates/edits files directly
Ask	Understand existing ABAP code, explain a BAPI, learn how GOS works, interpret a short dump

Switch with `/plan`, `/agent`, `/ask` slash commands.

What Bob cannot do for ABAP

Table: IBM Bob limitations for ABAP development

Limitation	Explanation
Cannot transport to SAP	Bob has no RFC/BAPI connection to SAP systems — code must be manually uploaded via SE38 or abapGit
Cannot activate objects	Activation happens in SAP workbench only
Cannot run ABAP	No SAP runtime — logic must be verified in the system
No SE80 GUI control	Cannot create screens (SE51), GUI status (SE41), or text elements graphically
Dictionary objects	Cannot create/activate DDIC objects (tables, domains, data elements) directly

Summary

Bob is a full-spectrum ABAP development partner — from generating complex programs like this `ZPO_DETAILS_DISPLAY` ALV report from a single prompt, to surgical fixes on existing code, to explaining legacy procedures and enforcing your team's coding standards. The only boundary is the SAP runtime itself — everything that can be authored as text (programs, classes, CDS views, FMs) Bob can generate, refactor, and review.

Generating ABAP code using IBM Bob and with extensions

This section describes the procedures for generating ABAP code using two extensions available within IBM Bob: the **ABAP FS Extension** and the **ABAP ADT (ABAP Development Tools) Extension**. Both extensions operate within the Visual Studio Code environment and enable developers to generate ABAP code through IBM Bob by providing natural language prompts and connecting to SAP source systems.

Generating ABAP code using the ABAP ADT extension

The ABAP ADT Extension enables developers to connect to an SAP source system directly through IBM Bob, create ABAP objects within a designated package, and generate ABAP code through IBM Bob by supplying a detailed natural language prompt. The extension additionally supports code activation within the connected system.

Procedure

Step 1: Install the extension into IBM Bob

Install the ABAP ADT Extension into IBM Bob.

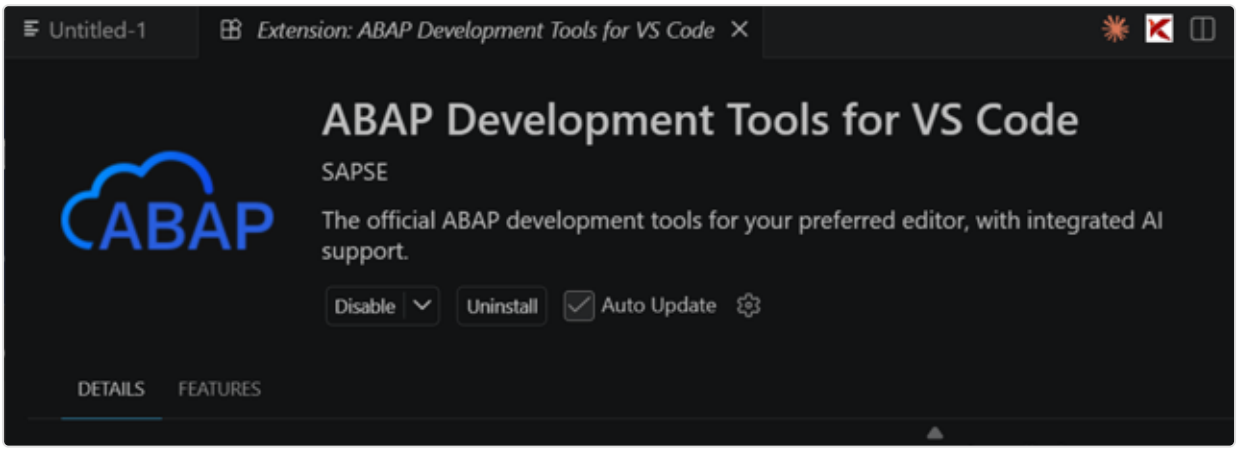


Figure: ABAP ADT Extension installed and active in IBM Bob

Step 2: Define the requirement prompt

Formulate a detailed prompt describing the ABAP development requirement. The following is an example of a prompt as provided in the source document:

```
Prompt: Generate a report to display the purchase orders and associated good receipt and invoices in an AL
```

Step 3: Connect to the source system

Establish a connection to the SAP source system by performing the following sub-steps:

- **Step 3a**: Create a new destination to the source system in IBM Bob by executing the command **Shift+Ctrl+P**.

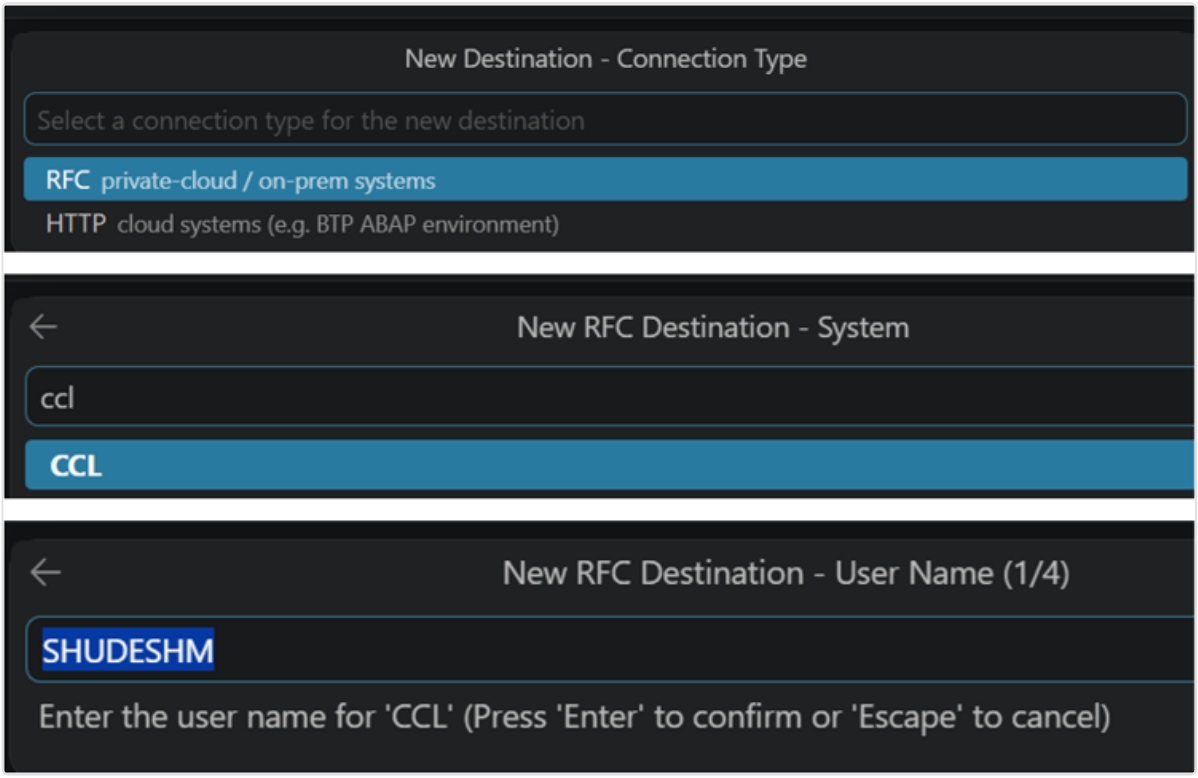


Figure: IBM Bob destination configuration dialog for connecting to an SAP source system

- **Step 3b**: Enter client of the system to be connected and destination name.

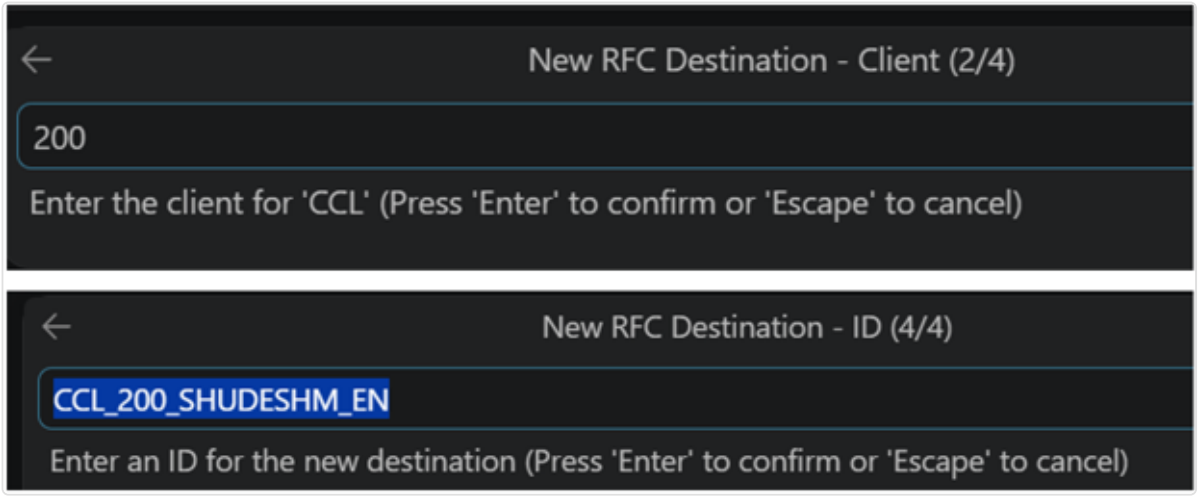


Figure: Entering the SAP system client number and destination name in IBM Bob

- **Step 3c**: On creating the destination, connect to the source system.

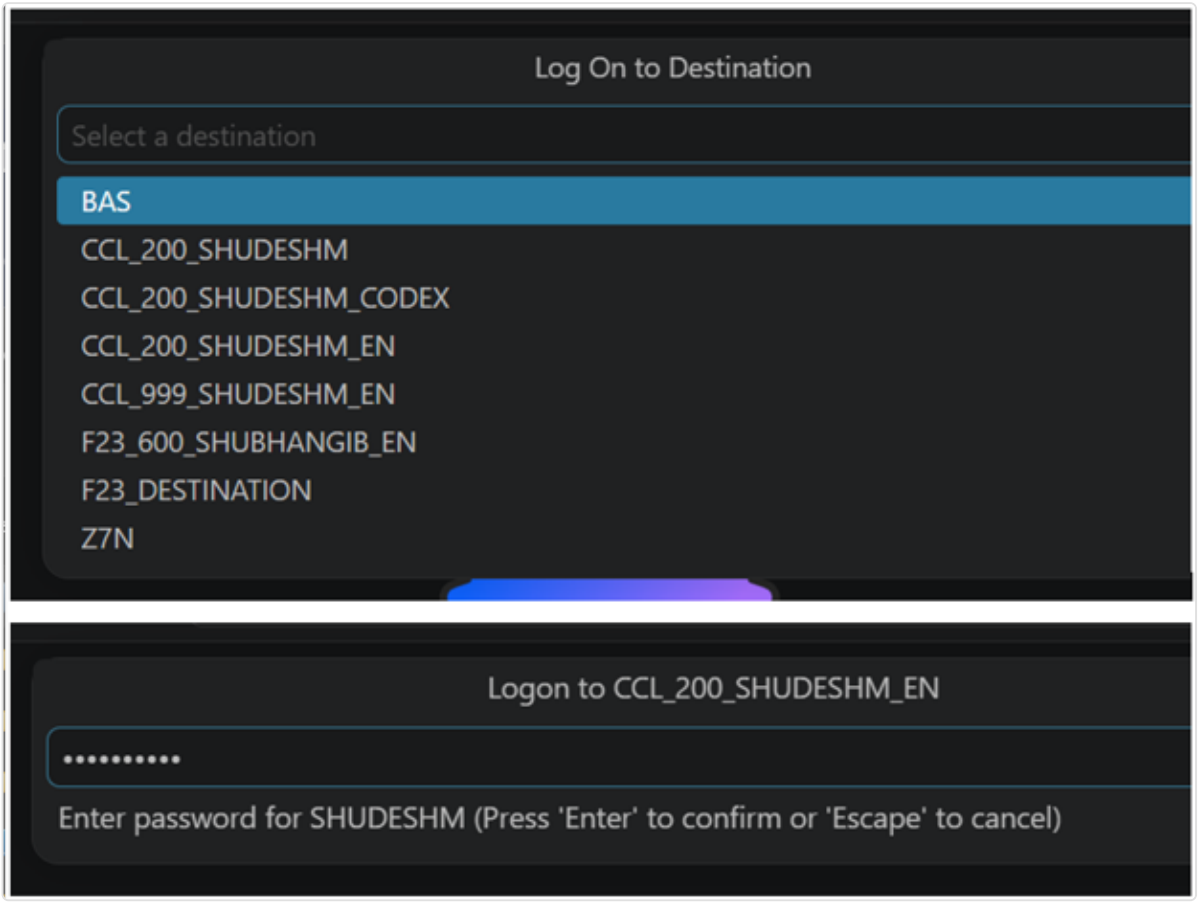


Figure: Successful SAP system connection established in IBM Bob via the ABAP ADT Extension

Step 4: Add the destination as a folder in the workspace

Add the destination as a folder to the workspace in IBM Bob.

Step 5: Add a package as a folder in the workspace

Add the target package as a folder in the workspace. This determines the package into which the newly created objects will be placed.

Step 6: Create a new object

Create a new ABAP object within the selected package.

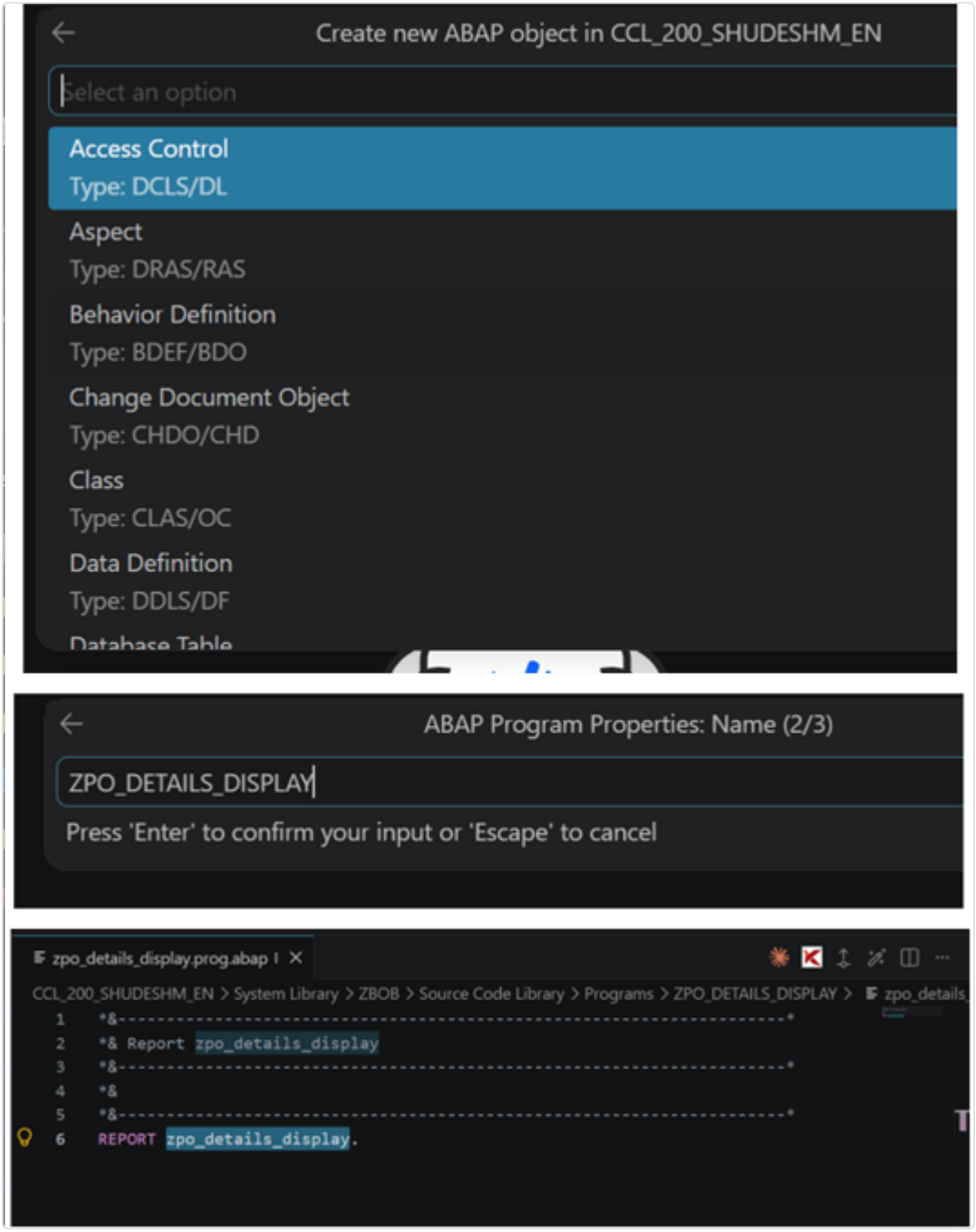


Figure: Creating a new ABAP object with the ABAP ADT Extension in IBM Bob

The newly created object can be confirmed as visible in the SAP system.

Step 7: Provide a prompt to IBM Bob to generate the code

Provide a prompt to IBM Bob to generate the ABAP code for the defined requirement.

Step 8: Activate the code

Activate the code. The code will be activated upon completion of this step.

Step 9: Paste the code in the system

Paste the generated code into the SAP system.

Additional capabilities

Following code generation, IBM Bob provides the following additional capabilities:

- **Explain the code**: IBM Bob can explain the generated ABAP code.
- **Improve the code**: IBM Bob can improve the generated ABAP code.
- **Contextualize the code**: IBM Bob can provide context for the generated ABAP code.

Generating ABAP code using the ABAP remote FS extension

The ABAP remote FS Extension enables developers to connect to an SAP system, create new ABAP objects, and generate ABAP code through IBM Bob by supplying a detailed prompt describing the requirement. The generated code must then be manually transferred to the SAP source system.

Procedure

Step 1: Install the extension (ABAP remote filesystem) into IBM Bob

Refer to the following diagram.

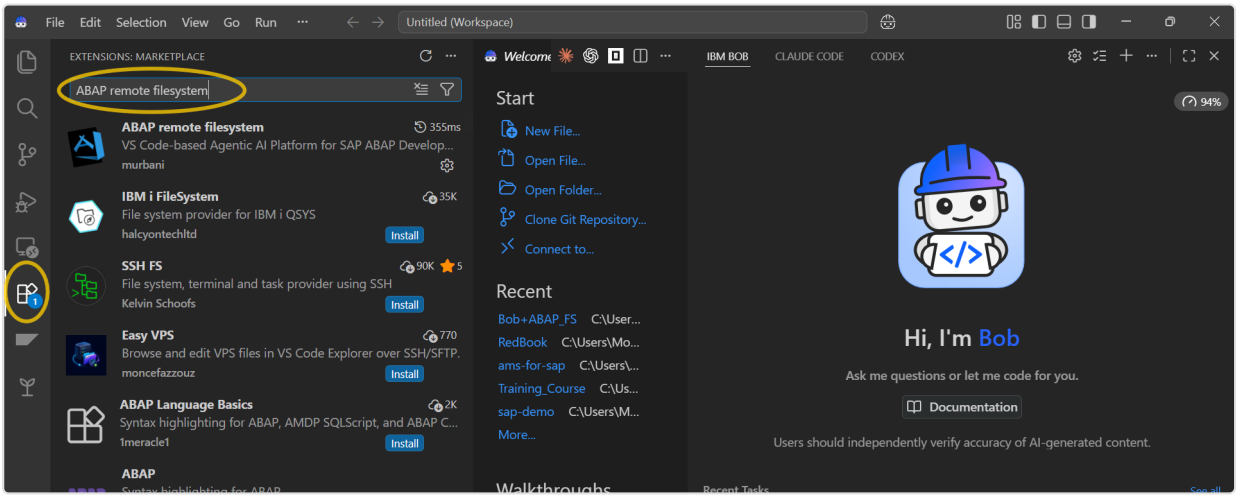


Figure: Searching for the ABAP remote filesystem extension in IBM Bob

Step 2.1: Accessing the ABAP FS Connection Manager

Refer to the following diagram.

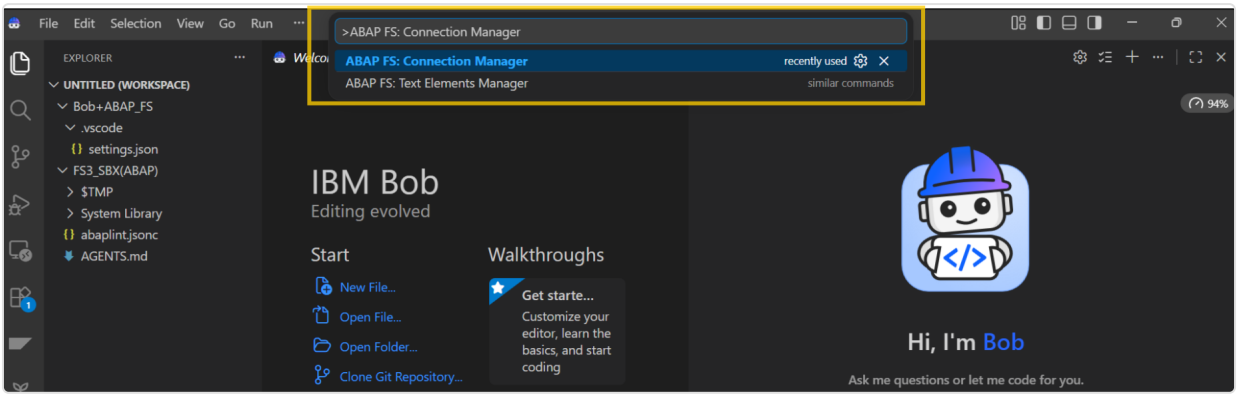


Figure: Opening the ABAP FS Connection Manager from the Command Palette

Step 2.2: Configure SAP connection into IBM Bob

Refer to the following diagram.

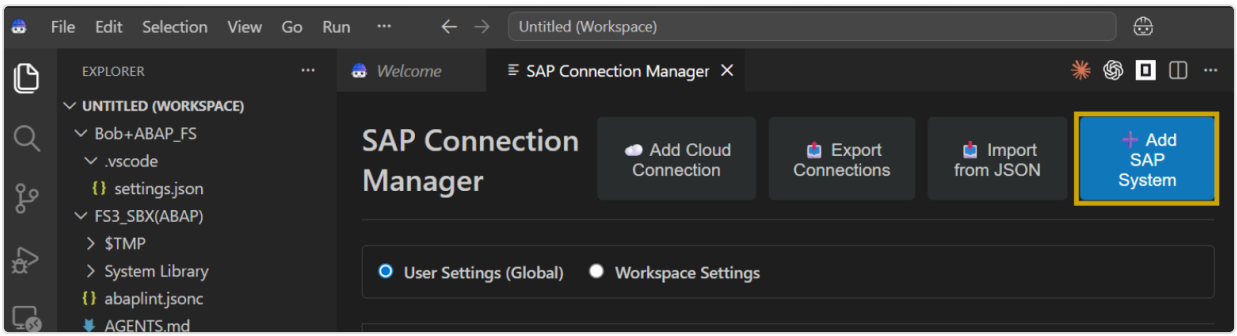


Figure: SAP Connection Manager with Add SAP System highlighted

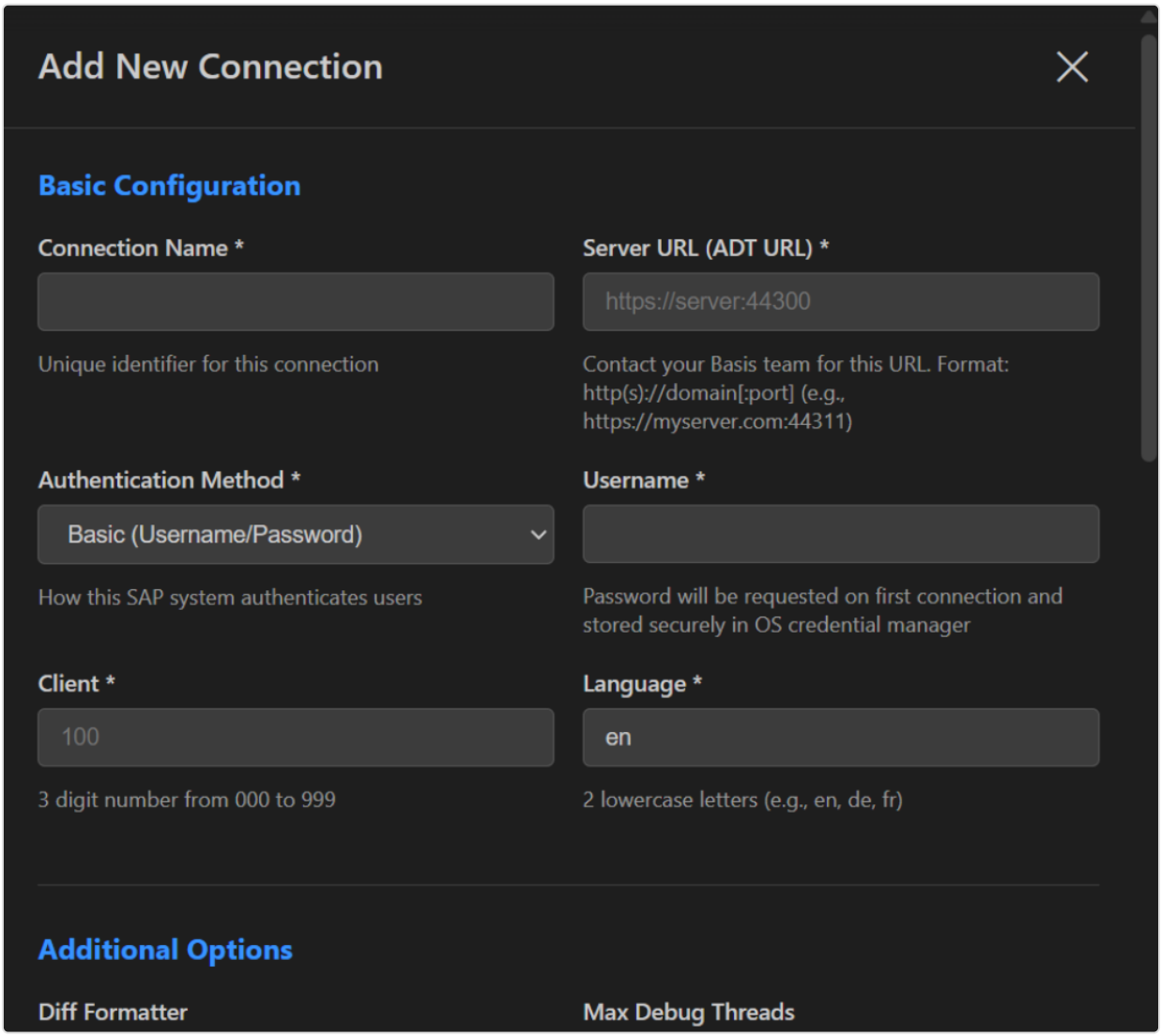


Figure: ABAP FS Add New Connection dialog

Step 3: Connect to the SAP system

Connect to the SAP system directly from IBM Bob using the ABAP FS Extension. As shown in the diagram below, execute the specified command to initiate the connection. When prompted, enter your SAP system password to establish access to the SAP system.

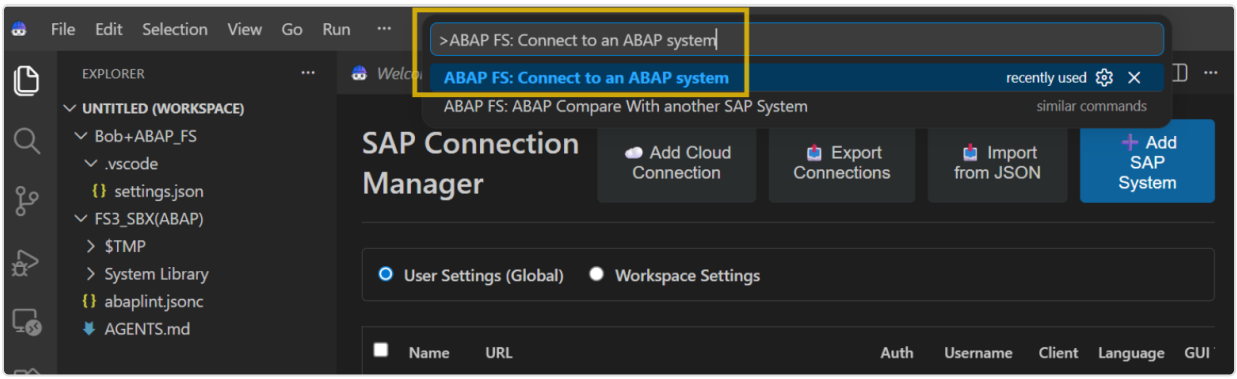


Figure: Connecting to an ABAP system from the Command Palette

Step 3.1: Connect to the SAP system

At the same time, the ABAP Remote FS Extension provides a built-in MCP server, which must be started by executing the command shown in the figure below. This MCP enables the generated ABAP program to be saved directly to the connected SAP system and activated.

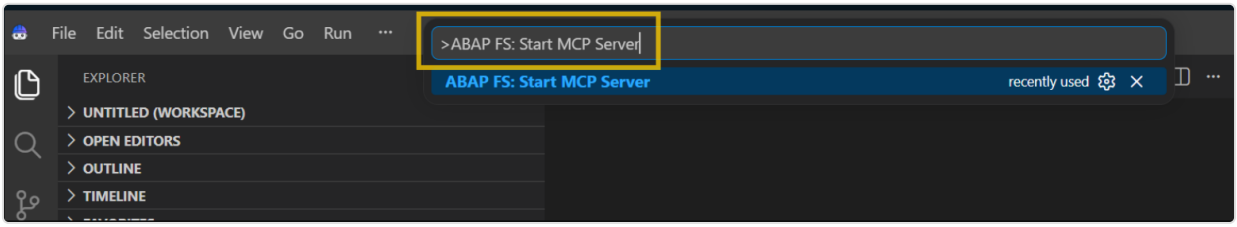


Figure: Starting the ABAP FS MCP server from the Command Palette

Step 4: Create a new object

Approach 1: Create a new ABAP object directly in the connected SAP system.

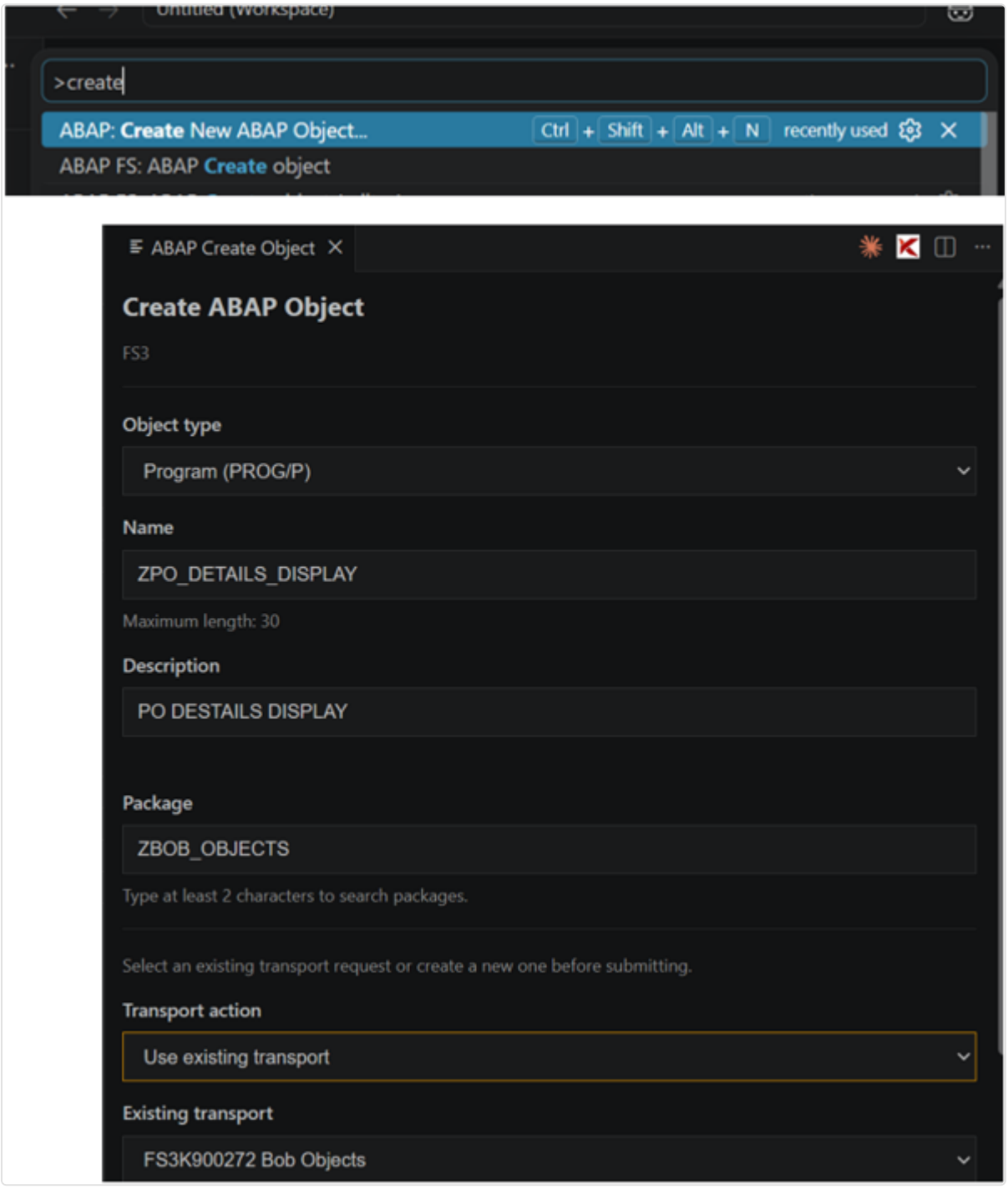


Figure: Creating a new ABAP object with the ABAP FS Extension in IBM Bob

Step 5: Provide a detailed prompt for the requirement

Approach 2: Provide a detailed prompt to IBM Bob describing the requirement for which ABAP code is to be generated.

> Create Version 1 (V1) of a Sales Order Reconciliation ABAP application with the basic functionality required for the initial release.

> **Object Details**
>
> - **Program Name:** `ZSO_RECONC`
> - **Description:** Sales Order Reconciliation
> - **Package:** `ZBOB4SAP`

Step 6: IBM Bob generates the code

IBM Bob generates the ABAP code based on the provided prompt.

Step 7: Activate the generated code into the SAP source system

The ABAP remote FS Extension uses its built-in MCP capabilities to save the generated ABAP code to the connected SAP system and activate the object.

IBM Bob for SAP BTP UI5 development using CAP, Node.js, and JavaScript

IBM Bob supports SAP BTP UI5 development built on the SAP Cloud Application Programming Model (CAP), Node.js, and JavaScript. It introduces CAP and SAP Business Application Studio (BAS) as the foundational platform context, then documents two distinct, complementary integration patterns — Outside-In, in which IBM Bob runs locally and reaches into BAS through a custom MCP server, and Inside-Out, in which an IBM Bob shell is installed and run directly inside the BAS terminal. Configuration steps, tool inventories, representative use cases, and security guidance are provided for both. The section is written for a mixed audience of SAP developers, solution architects, project managers, and senior management.

Introduction and purpose

SAP BTP UI5 development increasingly relies on the SAP Cloud Application Programming Model (CAP) together with Node.js and JavaScript, developed and deployed through SAP Business Application Studio (BAS). This section explains how IBM Bob supports that development workflow, from requirements gathering through code generation, review, documentation, and deployment.

Because BAS is a cloud-hosted development environment rather than a local file system, connecting an external AI development tool such as IBM Bob to it requires a deliberate integration approach. This section documents two such approaches, evaluates the trade-offs between them, and provides a complete configuration and usage procedure for each.

Scope

This section covers IBM Bob's role in SAP BTP UI5 development using CAP, Node.js, and JavaScript, and the two supported integration patterns between IBM Bob and SAP Business Application Studio: Outside-In Integration (see the Category 1: Outside-In Integration section) and Inside-Out Integration (see the Category 2: Inside-Out Integration section). It assumes the reader is already familiar with the general SAP application landscape and the SAP-released-tooling-first principle introduced elsewhere in this publication.

Foundational concepts

SAP Cloud Application Programming Model (CAP)

The SAP Cloud Application Programming Model (CAP) is a framework of languages, libraries, and tools for building enterprise-grade services and applications. It supports Java (with Spring Boot), and JavaScript and TypeScript (with Node.js) — among the most widely adopted languages for enterprise development. CAP is recommended by the SAP BTP Developer's Guide, and it guides developers along a path of proven best practices with a wide range of out-of-the-box solutions to recurring development tasks.

CAP includes built-in support for multitenancy through the `@sap/cds-mtxs` package.

The @sap/cds-mtxs package includes the following features:

- Provides a set of services that implement multitenancy, feature toggles, and extensibility — the three capabilities the "MTX" name refers to.
- Offers a straightforward way to integrate dependent services into a CAP application, such as the SAP SaaS Provisioning Service, SAP Service Manager, and the User Account and Authentication Service.
- Includes out-of-the-box handlers for subscribe and unsubscribe events — for example, to manage SAP HANA database containers automatically as tenants subscribe and unsubscribe.
- Supports custom logic within subscribe and unsubscribe event handlers. Typical examples include creating a destination to connect to a business system such as SAP S/4HANA, assigning roles, configuring Cloud Foundry routes, defining Kyma API rules, or provisioning additional SAP BTP service instances.

SAP Business Application Studio (BAS)

SAP Business Application Studio is an SAP BTP service that provides a modern, cloud-hosted development environment purpose-built for SAP Intelligent Enterprise applications. It is based on Code – OSS, the open-source project underlying Visual Studio Code, and delivers a desktop-like development experience — command line, optimized editors, and integrated tooling — as a cloud service.

At the center of SAP Business Application Studio are dev spaces: isolated, cloud-hosted environments comparable to dedicated virtual machines, each preconfigured with the tools and runtimes appropriate to a specific business scenario, such as SAP Fiori, SAP S/4HANA extensions, workflow development, or mobile development. This removes most of the manual effort of setting up a development environment and lets a team develop, test, build, and run a solution either locally within the dev space or in the cloud.

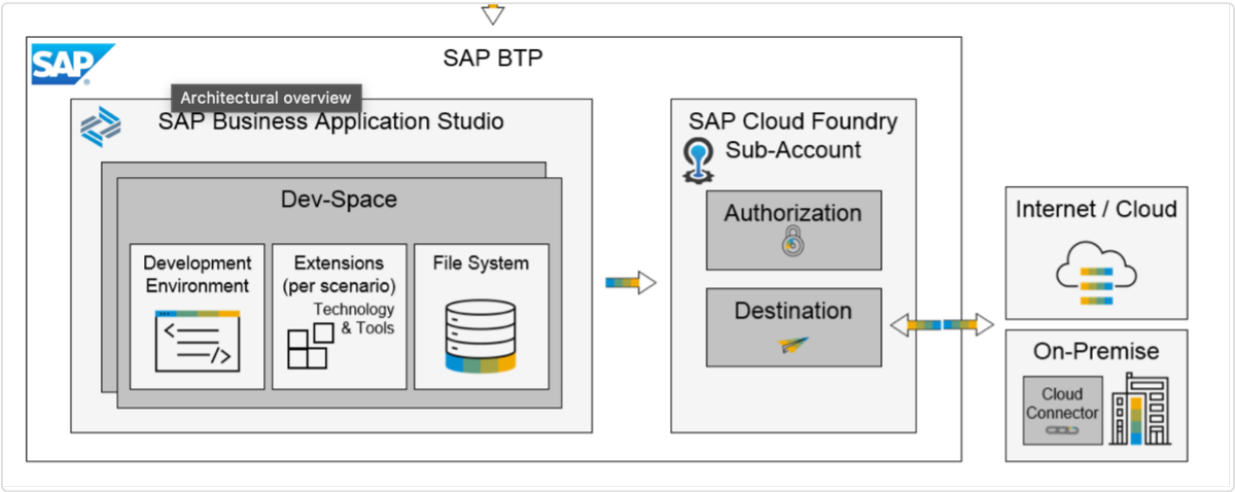


Figure: Architectural overview of SAP Business Application Studio within SAP BTP

This publication uses SAP Business Application Studio as the primary environment for developing CAP and SAPUI5 applications:

- SAP Business Application Studio is part of SAP BTP and is available on SAP's multi-cloud offering, spanning hyperscalers including AWS, Microsoft Azure, and Alibaba Cloud.
- Developers access their own dev space through a web browser. Each dev space is personal, access-restricted, and tailored to the developer's scenario — effectively a developer virtual appliance containing the development environment, technology stack, tooling, security and connectivity components, and the developer's own source code.
- SAP Business Application Studio is built on the SAP Cloud Foundry Business Application pattern, which provides multi-tenant software-as-a-service delivery.

- Identity and access management is based on SAP Cloud Foundry Authorization and Trust Management. Once a developer completes authorization, they are granted access to their dev space with the appropriate scopes.
- From within a dev space, developers can reach external services through SAP Cloud Foundry destinations. Where a service resides on-premise, the SAP Cloud Connector provides secured connectivity to that landscape.

Where IBM Bob fits in SAP BTP UI5 development

IBM Bob supports SAP BTP UI5 development across five broad categories of work:

1. **Requirement gathering**: creating and reviewing functional requirements
2. **Build work**: code generation for Node.js and JavaScript applications
3. **Review**: code review, performance review, and comment and documentation quality checks
4. **Reverse engineering**: producing documentation such as technical specifications directly from existing code
5. **Deployment**: assisting with application deployment tasks

Realizing these capabilities against a UI5 or CAP project that lives in SAP Business Application Studio requires establishing an integration between IBM Bob and BAS. The remainder of this section documents that integration in detail.

Two integration categories: Outside-In and Inside-Out

Integrating IBM Bob with SAP Business Application Studio can be approached in two distinct ways, differing in where IBM Bob itself runs and which side of the connection initiates control.

Table: Outside-In vs Inside-Out integration pattern comparison

Category 1 — Outside-In Integration	Category 2 — Inside-Out Integration
IBM Bob runs as a standalone installation on the developer's local machine.A custom MCP server, created inside the IBM Bob workspace, communicates with SAP BTP Business Application Studio using an authentication method such as Basic Authentication or OAuth 2.0.The UI5 project is downloaded into the local IBM Bob workspace from GitHub or GitLab.The developer interacts with IBM Bob locally, selecting and working on the UI5 project.	IBM Bob is not run as a standalone instance on the developer's local system.An IBM Bob shell is installed inside the SAP BTP BAS terminal, using a standard extension.An API key, created using the developer's registered IBM Bob user credentials (for IBM employees, their w3id and password), authenticates the shell.This API key is used to authenticate all communication from the BAS terminal to the IBM Bob service.

The following figure shows both patterns side by side: the Outside-In flow, in which IBM Bob on the developer's machine reaches into BTP through the BTP Administration MCP server and a dedicated BAS MCP server, and the Inside-Out flow, in which an IBM Bob shell runs natively inside the BAS dev space and authenticates outward to the IBM Bob service using an API key.

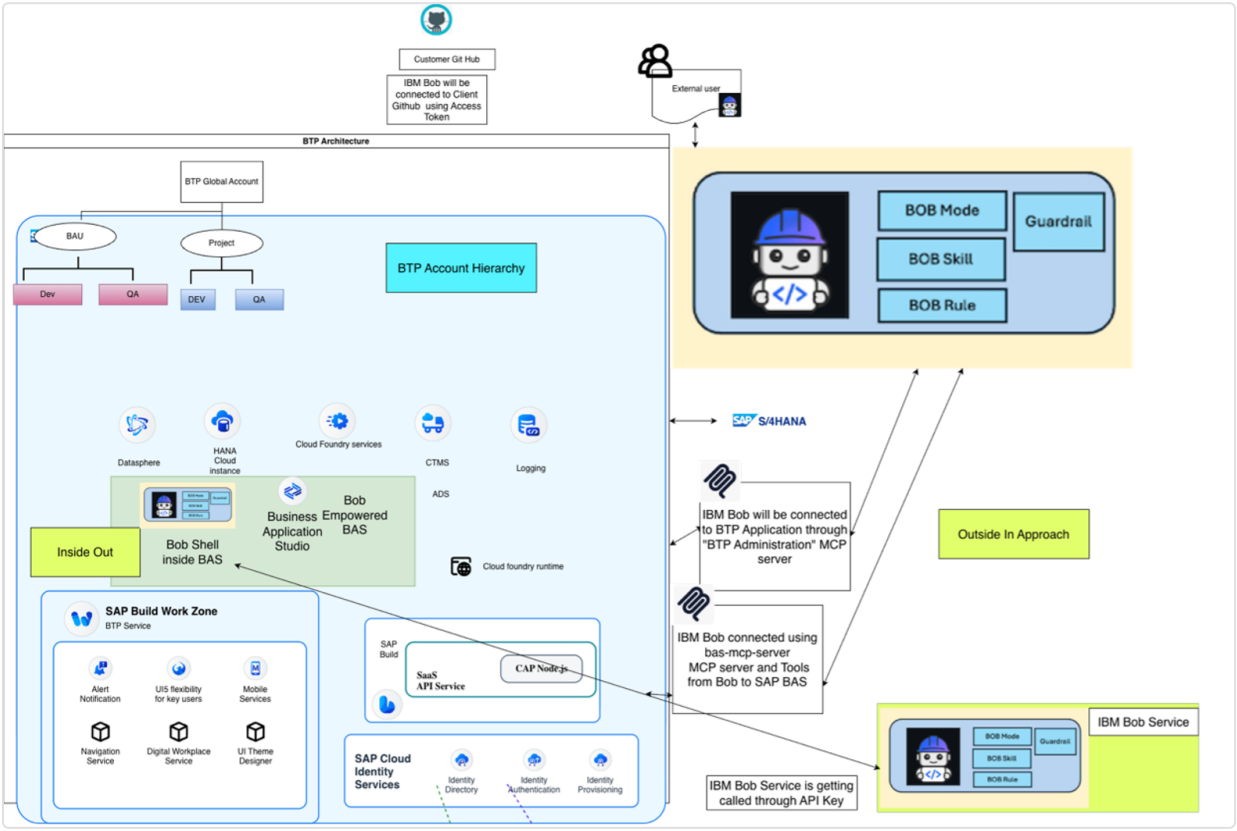


Figure: Outside-In and Inside-Out integration patterns for IBM Bob and SAP BAS

Category 1: Outside-In integration

This section describes a practical integration architecture in which IBM Bob, running as a standalone installation on a developer's local machine, connects to SAP Business Application Studio through a custom MCP server. The pattern treats BAS as an external system rather than a host environment, and it delivers the full range of AI-assisted development capabilities — code generation, refactoring, technical specification authoring, and unit test production — directly against UI5 projects that live in BAS.

Most AI-assisted development tools assume the AI runs inside the platform where the code lives. This section documents the opposite arrangement. IBM Bob runs locally; BAS runs in SAP's cloud; and a small, purpose-built MCP server bridges the two. That server — `bas-mcp-server` — authenticates to BAS using either Basic Authentication or OAuth 2.0, then exposes a set of file and command tools that IBM Bob treats as first-class capabilities.

The result is a developer workflow in which a UI5 project checked out from GitHub or GitLab into the local Bob workspace becomes the subject of a conversation. The developer selects a project, describes a task, and Bob uses the BAS connection to read source files, write generated artifacts back, run build commands, and search across the codebase — all without the developer leaving their local environment.

This section covers the architecture, the configuration procedure, all seven tools the server exposes, and five categories of use case that illustrate the range of what the integration makes possible.

The Outside-In architecture

What the name means

The phrase "outside-in" names the direction of control. In a conventional BAS-hosted AI workflow, the intelligence sits inside the BTP environment and acts on files that live in the same cloud. Here, the intelligence — IBM Bob — lives outside, on the developer's machine, and reaches into BAS through a structured API surface. The developer's local workspace is the point of control; BAS is the execution environment.

This distinction matters for two reasons. First, IBM Bob retains all of its local capabilities — custom modes, rule files, memory, and every configured MCP server — regardless of which BTP tenant the developer connects to. Second, the integration pattern works without any changes to BAS itself: BAS exposes a standard File API and a terminal execution endpoint, `bas-mcp-server` wraps those, and IBM Bob calls them through MCP. No BAS extension, no BTP subscription change, and no vendor dependency are required.

Component overview

The integration involves four components, each with a clearly defined role:

Table: Integration component summary.

Component	Location	Role
IBM Bob	Local machine	AI reasoning engine; sends tool calls; holds conversation context.
bas-mcp-server	Local machine (Node.js process)	MCP server; authenticates to BAS; translates tool calls into BAS File API requests.
SAP Business Application Studio	SAP BTP (cloud)	Hosts the dev space; exposes the File API and terminal execution endpoint.
UI5 project repository	GitHub / GitLab	Source of truth for project code; cloned into the local Bob workspace.

The `bas-mcp-server` process runs on the developer's machine alongside IBM Bob. It holds the BAS credentials — either a BTP email and password pair for Basic Authentication, or a pre-supplied OAuth 2.0 access token — and manages the token lifecycle automatically. IBM Bob's conversation engine never handles credentials directly; it issues tool calls and receives file content or command output in return.

Data flow

A typical interaction follows this sequence:

1. The developer clones a UI5 project from GitHub or GitLab into the local Bob workspace.
2. IBM Bob reads the local file structure to build context about the project.
3. The developer submits a prompt — for example, "generate a unit test for the `ProductList` controller."
4. IBM Bob issues a `bas_read_file` call to fetch the relevant source file from the live BAS dev space.
5. IBM Bob generates the artifact locally, then issues a `bas_write_file` call to write it back into BAS.
6. Optionally, IBM Bob issues a `bas_run_command` call — `npm run test`, for instance — and reports the result.

The local clone serves as the developer's reference copy and the source for IBM Bob's initial context-building. The BAS dev space serves as the live environment where generated files land and builds execute. The two remain in sync through the MCP bridge.

Authentication

Choosing an authentication mode

Two authentication modes are available. The choice is made once, at configuration time, by setting environment variables in the `bas-mcp-server` entry inside `~/.bob/settings/mcp.json`.

Table: Authentication mode selection.

Mode	Environment variables required	When to use
Basic Authentication (password grant)	BTP_EMAIL, BTP_PASSWORD	Development and trial environments, where the BTP user authenticates against accounts.sap.com.
OAuth 2.0 (pre-supplied token)	BAS_ACCESS_TOKEN	CI pipelines, shared environments, or any context where long-lived credentials must not be stored in plain text.

The server implements the XSUAA password-grant flow internally: it derives the token endpoint from `BAS_URL` by replacing `applicationstudio` with `authentication`, posts the credentials, caches the resulting token in memory, and refreshes automatically on a `401` response. When `BAS_ACCESS_TOKEN` is set, the server skips the grant flow entirely and uses the supplied token until it expires, then falls back to a refresh.

Basic authentication configuration

The `bas-mcp-server` entry in `~/.bob/settings/mcp.json` for Basic Authentication reads as follows:

```
"bas-mcp-server": {
"command": "/usr/local/bin/node",
"args": [
"/Users/<your-username>/Documents/IBM BOB/Workspace/bas-mcp-server/build/index.js"
],
"env": {
"BAS_URL": "https://<your-trial-subdomain>.applicationstudio.cloud.sap",
"BTP_EMAIL": "your.email@domain.com",
"BTP_PASSWORD": "<your-btp-password, stored in an environment variable>"
},
"alwaysAllow": ["bas_list_files", "bas_read_file"],
"disabled": false
}
```

> **Important:** Never commit `mcp.json` to version control while it contains plaintext credentials. Use an environment-variable reference or a secrets manager, and substitute the value at runtime.

OAuth 2.0 configuration

For environments where a token is obtained externally — through a CI pipeline, a service-account token rotation script, or a corporate identity broker — supply the token directly:

```
"bas-mcp-server": {

"command": "/usr/local/bin/node",

"args": [

"/Users/<your-username>/Documents/IBM BOB/Workspace/bas-mcp-server/build/index.js"

],

"env": {

"BAS_URL": "https://<your-subdomain>.applicationstudio.cloud.sap",

"BAS_ACCESS_TOKEN": "<oauth2-access-token>"

},

"alwaysAllow": ["bas_list_files", "bas_read_file"],

"disabled": false

}
```

The token lifetime is honored: the server adds `expires_in` to the current timestamp and attempts a refresh when the token has less than sixty seconds remaining. If no refresh credentials are available and the token expires, the server returns a `401` error to IBM Bob, which surfaces it in the conversation as an actionable message.

Setting up the local workspace with a UI5 project

Before any AI-assisted task can begin, the UI5 project must be available locally. Clone it from GitHub or GitLab into the Bob workspace:

```
*# Clone from GitHub*

cd "/Users/<your-username>/Documents/IBM BOB/Workspace"

git clone https://github.com/<org>/<ui5-project>.git

*# Or from GitLab*

git clone https://gitlab.<your-domain>/<group>/<ui5-project>.git
```

Once cloned, IBM Bob reads the local directory at the start of each conversation. The developer opens IBM Bob, navigates to the project folder, and the workspace context is established automatically. From that point, every prompt IBM Bob receives is grounded in the project's actual source code.

The BAS MCP server tool inventory

The `bas-mcp-server` exposes seven tools over the MCP stdio transport. Each wraps one or more calls to the BAS File API or terminal execution endpoint. The table below lists them with their risk profiles, following the same suffix convention as the BTP Administration MCP server described elsewhere in this publication.

Table: BAS MCP server tool inventory.

Tool	Operation	Risk profile
bas_list_files	List files and folders in a directory	Read-only
bas_read_file	Read the content of a single file	Read-only
bas_write_file	Create or overwrite a file	Write — verify path before use
bas_delete_file	Delete a file or empty directory	Destructive — irreversible
bas_create_folder	Create a new directory	Write
bas_search_files	Search for a text pattern across files	Read-only
bas_run_command	Execute a shell command in the BAS terminal	Execute — confirm command before use

`bas_list_files` and `bas_read_file` are marked as `alwaysAllow` in the default configuration, meaning IBM Bob invokes them without confirmation. The write and execute tools require confirmation unless the developer explicitly adds them to `alwaysAllow` — a deliberate design decision that prevents unreviewed file changes from landing in the dev space during exploratory conversations.

Use cases

Use case 1: Code generation

Scenario: A developer needs a new SAPUI5 controller for a purchase order list view. The project already contains several controllers, and the new one must follow the same structure and naming conventions.

IBM Bob reads the existing controllers using `bas_read_file` to establish the project's patterns — namespace declarations, lifecycle method signatures, formatter references, and i18n key conventions. It then generates the new controller in full, writes it to the correct path with `bas_write_file`, and creates the matching view XML file alongside it.

```
Developer: "Generate a PurchaseOrderList controller following the same

pattern as the existing SalesOrderList controller."
```

IBM Bob calls `bas_read_file` on `SalesOrderList.controller.js`, constructs the new controller with all lifecycle methods, formatter references, and i18n keys adapted for purchase orders, then calls `bas_write_file` to place it at `webapp/controller/PurchaseOrderList.controller.js`. A second `bas_write_file` call creates `webapp/view/PurchaseOrderList.view.xml`. The developer reviews the diff locally in the cloned repository, commits, and pushes.

Use case 2: Code refactoring

Scenario: The project has accumulated several controllers that mix data-fetching logic with UI event handling. The team has decided to extract all OData calls into a dedicated service layer.

This use case demonstrates the full read-transform-write cycle. IBM Bob uses `bas_search_files` to locate every file that contains direct `this.getModel().read()` or `this.getModel().create()` calls. For each match, it reads the file, extracts the OData calls, relocates them to a service module, and rewrites the calling controller to use the new abstraction.

```
Developer: "Refactor all direct OData calls across the project into

a centralized ODataService.js module."
```

IBM Bob issues `bas_search_files` with `query: "this.getModel().read"` and `include: "*.js"`, receives the list of affected files, reads each one, produces the refactored versions and the new service module, then writes all of them back in a single conversation turn. It reports what changed and what remains for manual review — particularly any dynamic query parameters that require human judgment.

> **Tip:** For large refactoring tasks, ask IBM Bob to process one controller at a time and review each before proceeding. This keeps the conversation context manageable and the diffs reviewable.

Use case 3: Technical specification generation

Scenario: A project manager needs a technical specification document for a new UI5 application feature, to be shared with the offshore development team. The specification must describe the component structure, the OData service bindings, the navigation flows, and the error handling strategy.

IBM Bob reads the relevant source files — the manifest, the views, the controllers, and any existing annotations — and produces a structured specification document. This use case does not write back to BAS; it generates a local Markdown file in the Bob workspace that the project manager can export and share.

```
Developer: "Generate a technical specification for the SupplierInvoice
feature covering component structure, OData bindings, navigation flows and error handling."
```

IBM Bob calls `bas_list_files` on `webapp/`, then reads `manifest.json`, the relevant view and controller files, and any `.annotations.xml` files. It assembles a specification with sections for component overview, data model, navigation model, binding strategy, error states, and open questions. The specification lands in the local workspace as `docs/spec-SupplierInvoice.md`.

This pattern is particularly useful when a feature exists in BAS but has never been formally documented. IBM Bob reads what is actually there rather than what someone remembers, so the specification reflects the code's real behavior.

Use case 4: Unit test generation

Scenario: The team has adopted a policy requiring unit test coverage for all controller methods that perform data manipulation. Several existing controllers have no tests, and the developer needs to bring them into compliance before the sprint closes.

IBM Bob reads the controller under test, identifies the public methods that manipulate data or model state, generates QUnit test cases for each one — including positive paths, negative paths, and boundary conditions — and writes the test file to the project's `test/unit/` directory. It then runs the test suite using `bas_run_command` and reports the results.

```
Developer: "Generate QUnit unit tests for ApprovalWorkflow.controller.js covering all methods that modify
generating them."
```

IBM Bob reads `ApprovalWorkflow.controller.js`, identifies five methods — `onApprove`, `onReject`, `onDelegate`, `_validateComment`, and `_loadWorkflowItems` — and generates a test file with twenty-three test cases. It writes the file to `test/unit/controller/ApprovalWorkflow.qunit.js`, then issues:

```
cd projects/my-ui5-app && npm run test:unit
```

The test results come back through `bas_run_command`. IBM Bob surfaces the pass/fail counts and flags any tests that failed, with an explanation of the likely cause and a suggested fix.

> **Important:** Generated tests require review before merging. IBM Bob produces structurally correct test code, but the expected values in assertions depend on assumptions about business logic. Confirm that boundary conditions reflect actual requirements.

Use case 5: Cross-file analysis and documentation

Scenario: A new team member needs to understand the data flow for a complex feature — specifically, how a user action in a list view triggers an OData batch operation and updates three separate models. No existing documentation covers this flow.

IBM Bob uses `bas_search_files` to trace the call chain from the initial button-press event through to the batch call. It reads each file in the chain — controller, formatter, service module, and any relevant annotations — and produces a written explanation of the data flow, complete with a Mermaid sequence diagram showing the component interactions.

```
Developer: "Explain the complete data flow when a user clicks 'Submit
for Approval' in the PurchaseOrderList view, including all OData calls
and model updates."
```

IBM Bob traces the `onSubmitForApproval` event handler through four files, maps the sequence of model reads, OData batch calls, and success and error callbacks, and writes a `docs/data-flow-SubmitForApproval.md` file to the local workspace. The document includes the sequence diagram, a table of all OData operations invoked, and a list of UI state changes.

This use case has no write-back to BAS. Its value is purely analytical — accelerating onboarding, supporting code reviews, and producing documentation that would otherwise require a senior developer to sit with a whiteboard for an hour.

Configuring the full integration

Prerequisites

Before starting, confirm that the following are in place:

- IBM Bob is installed and running on the local machine. Version 2.0.5 or later is recommended; verify with `bob --version` in the terminal.
- Node.js 18 or later is installed at `/usr/local/bin/node`.
- The `bas-mcp-server` source has been cloned into the Bob workspace and built (see the commands below).
- Access to a SAP BAS dev space is available, with at least one UI5 project present.
- The UI5 project has been cloned locally from GitHub or GitLab into the Bob workspace.

```
cd "/Users/<your-username>/Documents/IBM BOB/Workspace/bas-mcp-server"

npm install

npm run build
```

End-to-end configuration steps

Step 1: Build the server

Compile the TypeScript source to produce the Node.js bundle that IBM Bob will invoke:

```
cd "/Users/<your-username>/Documents/IBM BOB/Workspace/bas-mcp-server"

npm install && npm run build

*# Expected output: build/index.js*
```

Step 2: Register the server in Bob's global MCP configuration

Open `~/.bob/settings/mcp.json` and add the `bas-mcp-server` entry under `mcpServers`, using the Basic Authentication template from the Basic Authentication Configuration section or the OAuth 2.0 template from the OAuth 2.0 Configuration section, depending on the target environment.

Step 3: Verify the registration

Open Bob Settings → MCP. The `bas-mcp-server` entry should appear with a connected indicator. If it shows an error, check the Node.js path and confirm that `build/index.js` exists.

Step 4: Confirm tool availability

In a Bob conversation, ask the question below. Bob should list all seven tools; if only `bas_list_files` and `bas_read_file` appear, the remaining tools are present but not pre-approved and will require a one-time confirmation per session unless added to `alwaysAllow`.

```
Developer: "What tools do you have available from the bas-mcp-server?"
```

Step 5: Open the UI5 project workspace

Navigate Bob's workspace to the cloned UI5 project directory. Bob's context window populates with the project's file structure on the next conversation start.

Step 6: Submit a verification prompt

Confirm the integration end-to-end with a read-only test:

```
Developer: "List the files in the webapp/controller directory of my BAS dev space."
```

IBM Bob calls `bas_list_files` and returns the directory contents. If this succeeds, the full integration is operational.

Security best practices

For security, consider the following precautions:

- **Do not commit credentials.** The `~/.bob/settings/mcp.json` file lives outside all version-controlled directories by design. Keep it there, and never copy it into a project repository.

- **Restrict `alwaysAllow` to read-only tools in shared environments.** Where multiple developers share a BAS dev space, limit pre-approved tools to `bas_list_files` and `bas_read_file`; write and execute operations should require explicit confirmation.
- **Rotate BTP passwords on the same schedule as other privileged accounts.** A password stored in a configuration file that persists on a developer's machine for months creates a standing exposure; apply the organization's standard rotation cadence for service accounts.

Operational patterns and limits

Keeping the local clone in sync

The local Git clone and the BAS dev space can diverge if files are written to BAS through the MCP server without a corresponding commit and push. Establish a working practice: after any session in which IBM Bob has written files to BAS, pull those changes into the local clone before starting the next session.

```
cd "/Users/<your-username>/Documents/IBM BOB/Workspace/<ui5-project>"

git pull origin <branch>
```

This preserves the local clone as a reliable reference for IBM Bob's context-building and prevents duplicate work across sessions.

Context window management for large projects

A UI5 project with dozens of controllers and views can produce a substantial file listing. IBM Bob reads what it needs rather than loading every file at once, but a prompt such as "refactor the whole project" generates many sequential tool calls and may approach the conversation context limit before finishing.

The practical mitigation is to scope tasks explicitly. "Refactor the OData calls in the three Approval controllers" outperforms "refactor all OData calls in the project" in both speed and output quality. IBM Bob works best when the scope matches what a skilled developer would tackle in a single focused session.

Tool confirmation in practice

The `bas_write_file`, `bas_delete_file`, and `bas_run_command` tools require confirmation by default. This is not friction to be engineered around: the confirmation prompt is the developer's last opportunity to review what IBM Bob is about to do before it happens. Read the proposed file content or command before approving. For `bas_run_command` in particular, a misread prompt could trigger a destructive operation — `rm -rf`, for example, is syntactically valid and will succeed without further warning.

Category 2: Inside-Out integration

In the Inside-Out pattern, IBM Bob is not installed as a standalone instance on the developer's local machine. Instead, an IBM Bob shell is installed directly inside the SAP BTP Business Application Studio terminal, using a standard extension, and authenticates to the IBM Bob service using an API key generated from the developer's IBM Bob account. This section documents the two-part setup — generating the API key, then installing and configuring the Bob shell inside BAS — and how to work with it once it is ready.

Generating an IBM Bob API key

The API key is generated once, from the IBM Bob web portal, and is then supplied to the Bob shell running inside the BAS terminal.

Step 1: Sign in to the IBM Bob portal

Navigate to `bob.ibm.com` and log in to your account.

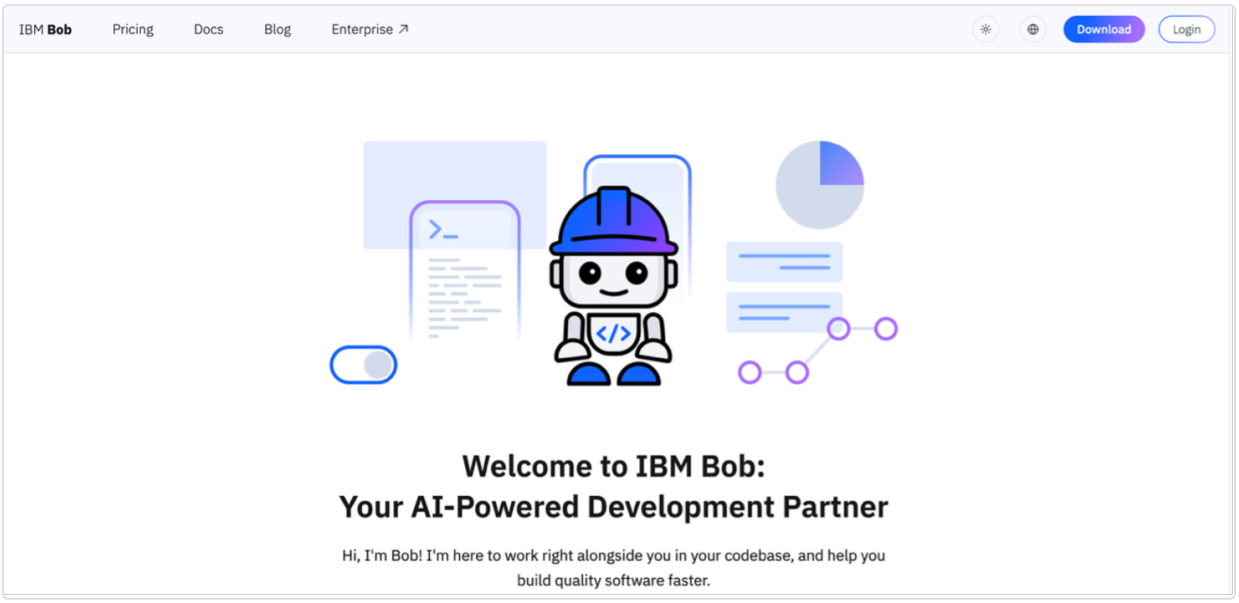


Figure: The IBM Bob portal landing page at bob.ibm.com

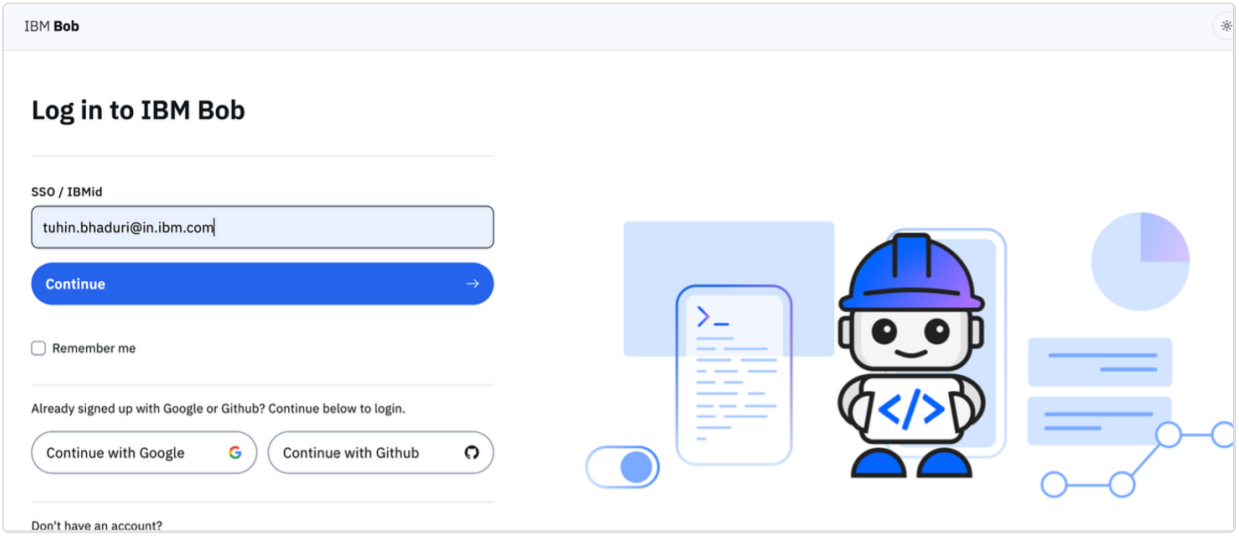


Figure: Signing in to IBM Bob using SSO / IBM ID

Step 2: Open your subscription instance

Select the applicable workspace, then locate the API Keys section in the left-hand navigation.

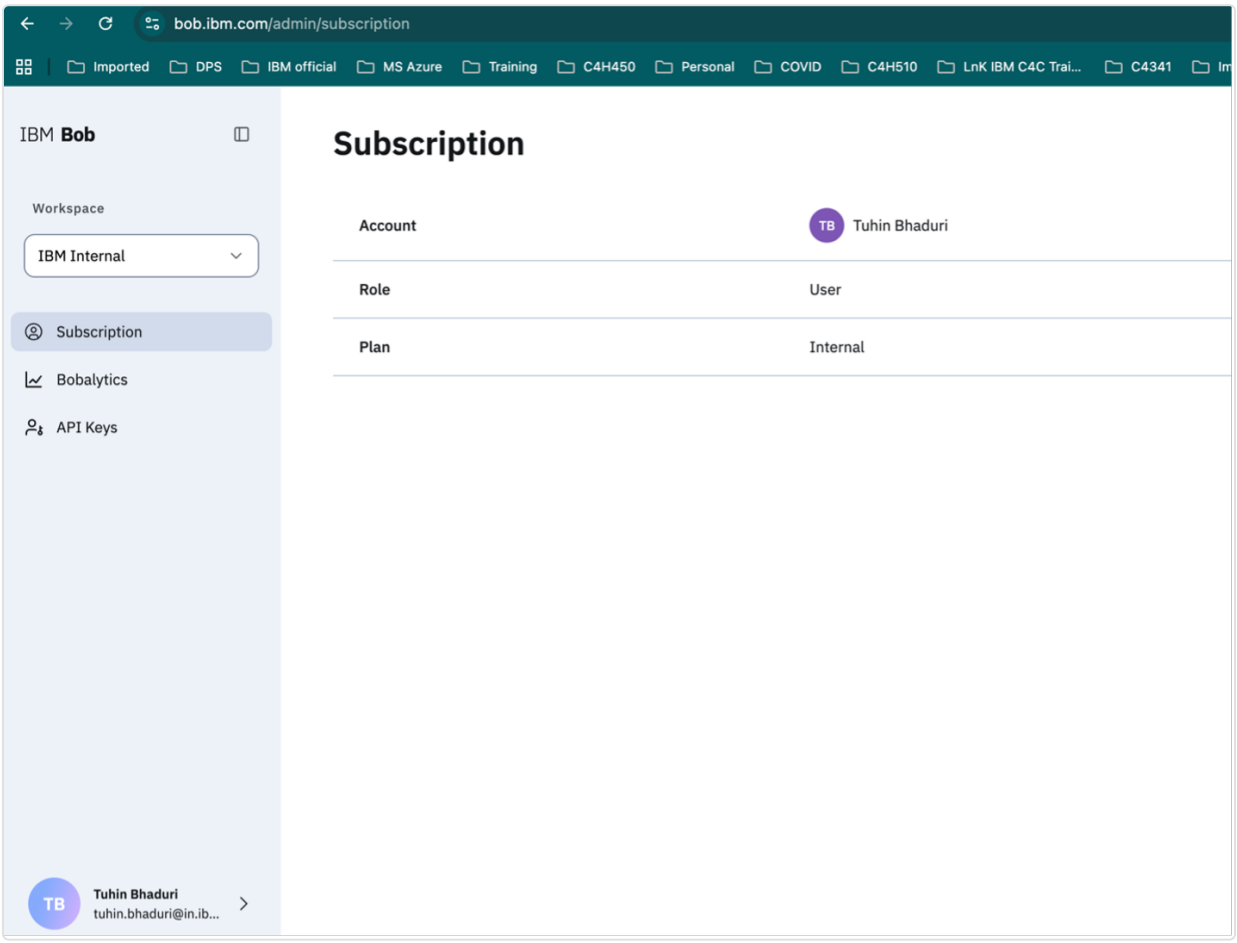


Figure: The Subscription page, showing the API Keys entry in the left-hand navigation

Step 3: Create a new API key

Select Create a new API key, provide a descriptive name and, optionally, a description, and choose the appropriate scope for the key.

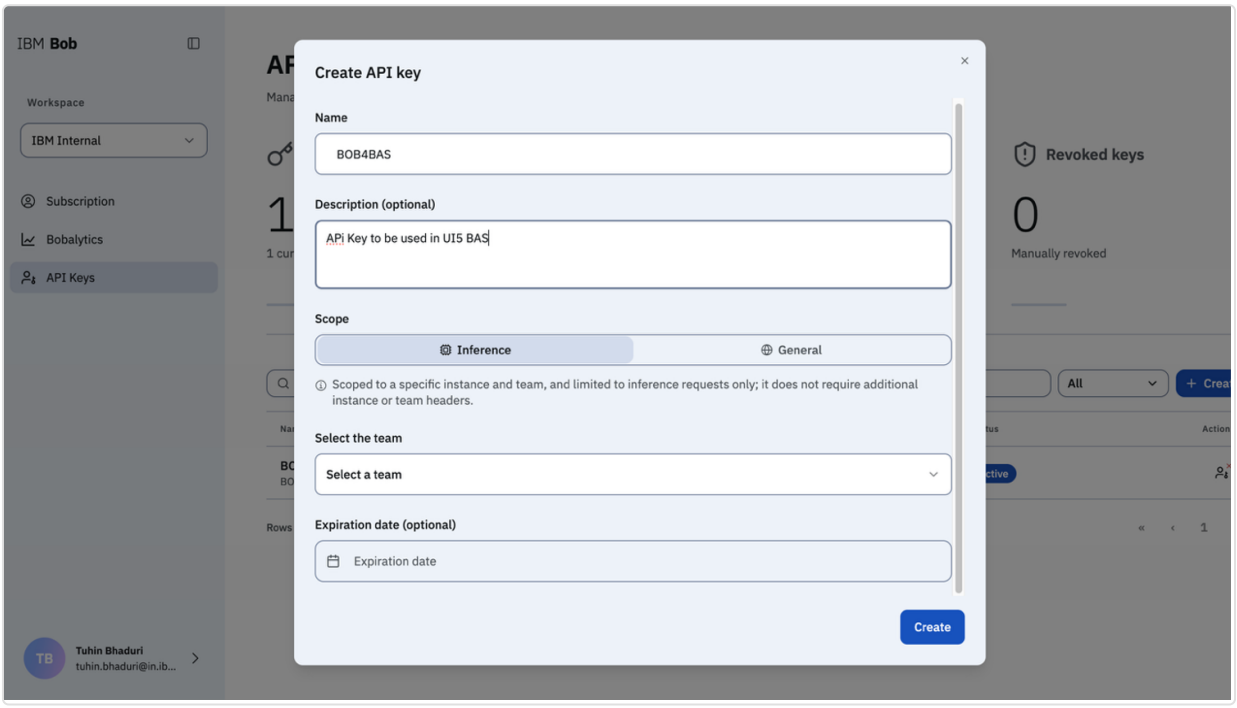


Figure: The Create API key dialog, including name, description, and scope options

Step 4: Save the key

Once created, the key appears in the API Keys list with a status of Active. Copy and store it securely — it will not be displayed again in full.

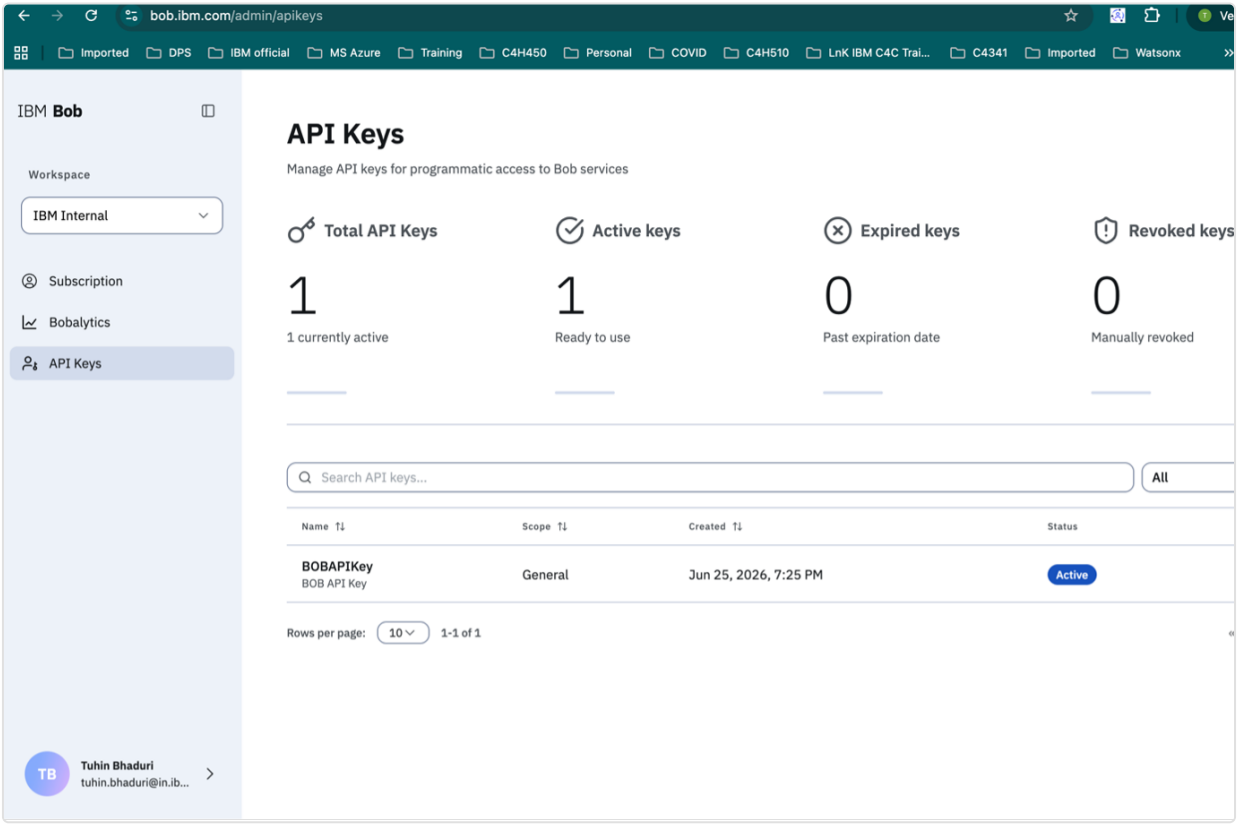


Figure: The API Keys list, showing a newly created and active key

> **Important:** Treat the IBM Bob API key as a privileged credential. It authenticates every request the Bob shell makes to the IBM Bob service on the developer's behalf. Do not share it, and store it only using the mechanisms described in the Security considerations for Inside-Out integration section.

Installing IBM Bob shell inside SAP BTP BAS

With the API key generated, the next step is to install the IBM Bob shell inside the BAS terminal.

Step 1: Open SAP BTP BAS

Sign in to SAP BTP Business Application Studio using your BTP user ID and credentials, and open a terminal within your dev space.

Step 2: Install the Bob shell

Run the following command in the BAS terminal:

```
curl -fsSL https://bob.ibm.com/download/bobshell.sh | bash
```

This installs the IBM Bob shell inside the BAS dev space. Node.js version 22.15 or later is required for the shell to run; the installer verifies this automatically.

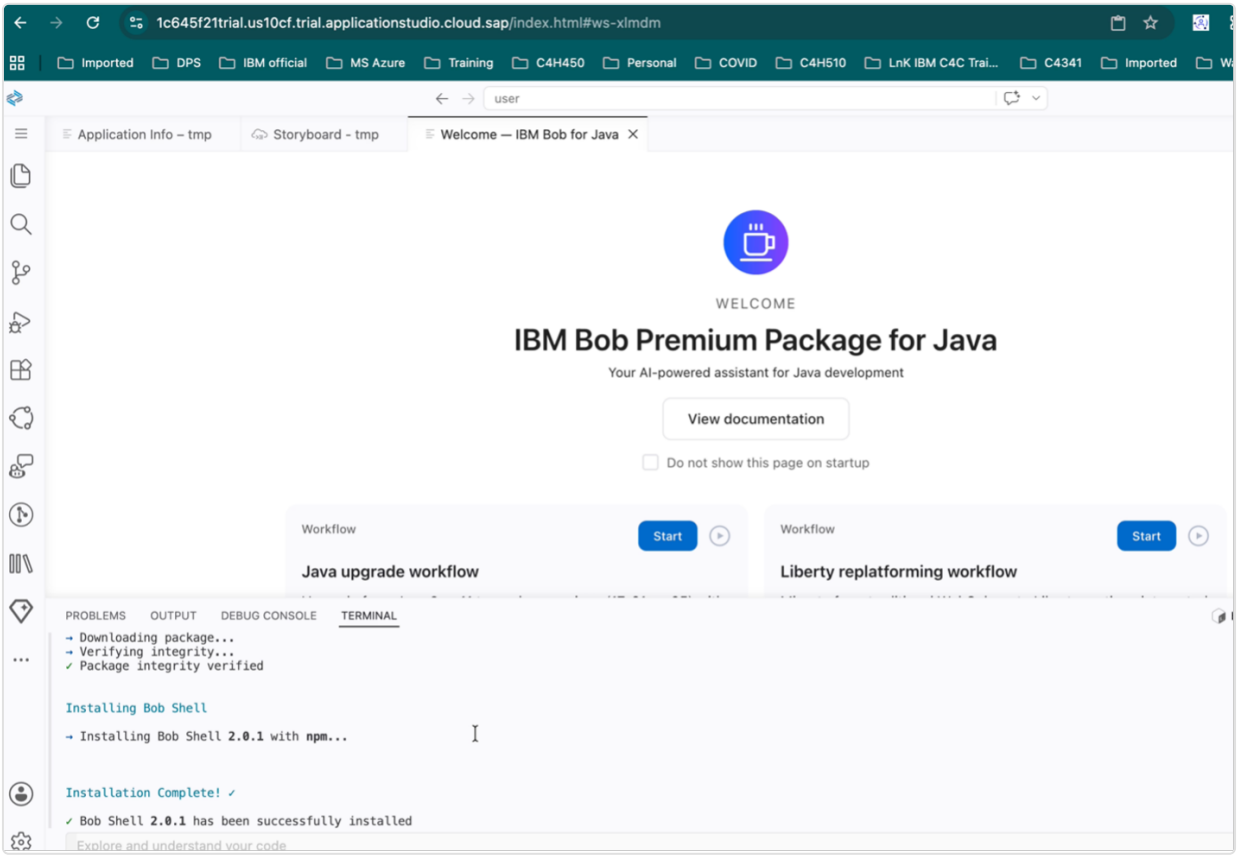


Figure: The BAS terminal downloading and installing the Bob Shell package

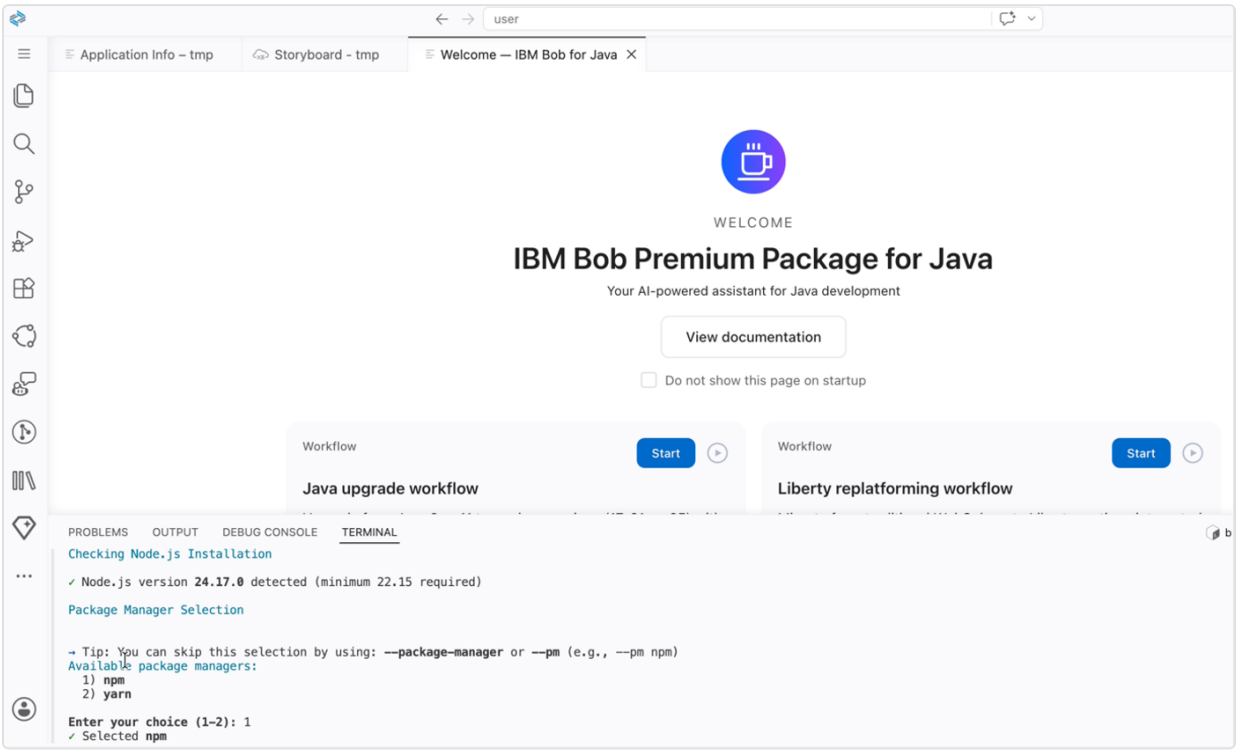


Figure: Bob Shell installation, Node.js check and package manager selection

Linking the API key to the Bob shell

Two commands are available to link the API key generated in the Generating an IBM Bob API Key section to the Bob shell session. Understanding the difference between them matters, since one is session-scoped and the other persists across sessions.

Table: API key linking commands — session-scoped vs persistent

Command	Behavior
export BOB_API_KEY="..."	Links the API key to the current Bob shell session only. The key is lost on logoff, and must be re-entered at the start of every new session.
echo 'export BOB_API_KEY="..."' >> ~/.bashrc && source ~/.bashrc	Persists the API key by writing it into the shell's source file. This avoids re-entering the key on every session, but stores it in a file inside the dev space, so the Security Considerations for Inside-Out Integration section should be reviewed before adopting this approach on a shared dev space.

Session-scoped linking:

```
export BOB_API_KEY="***Put your IBM Bob API key that starts with bob_prod_bob-***"
```

Persistent linking:

```
echo 'export BOB_API_KEY="***Put your IBM Bob API key that starts with bob_prod_bob-***"' >> \~/.bashrc

source \~/.bashrc
```

Once the linking is established, the IBM Bob shell inside BTP BAS is ready to use.

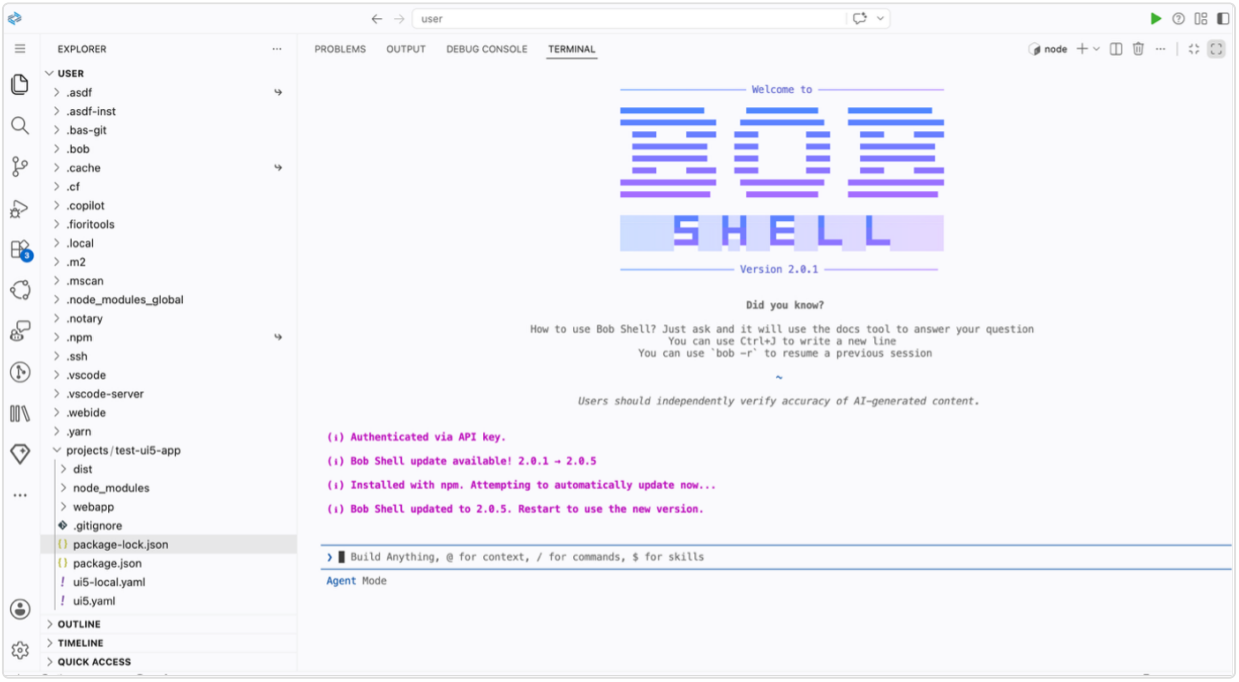


Figure: Bob Shell banner confirming installation and API key authentication

Using IBM Bob shell inside BAS

With the shell installed and authenticated, run the `bob` command in the BAS terminal to start a session.

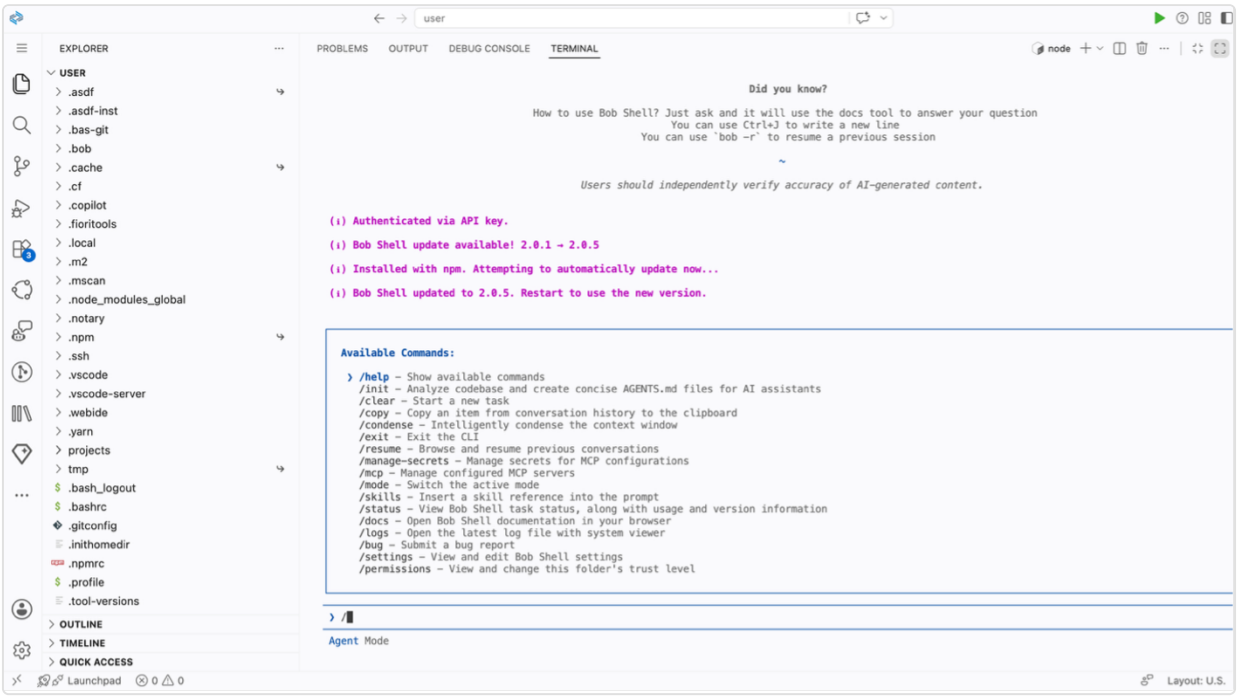


Figure: Bob Shell available commands, listed from within the BAS terminal

From this point, the developer can move Bob skills, Bob rules, and Bob modes from their local workspace into the Bob shell running inside BAS, or bring them in directly from the project's own repositories. Switching to a custom mode configured for UI5 development — for example, one that encodes the project's controller and view conventions — tailors the shell's behavior to the team's standards before development begins.

Security considerations for Inside-Out integration

Consider the following security aspects:

- **Treat the API key as a privileged, long-lived credential once persisted.** Writing `BOB_API_KEY` into `~/.bashrc` removes the need to re-enter it, but it then resides in plaintext inside the dev space's file system for as long as the dev space exists.
- **Avoid persisting the API key in shared or pooled dev spaces.** Where a dev space is used by more than one developer, prefer the session-scoped `export` command and accept the minor inconvenience of re-entering the key, rather than embedding it in a shared shell profile.
- **Rotate and revoke API keys through the IBM Bob portal** on the same cadence used for other service credentials, and revoke a key immediately if a dev space is decommissioned, reassigned, or suspected of compromise.
- **Confirm the Node.js version requirement (22.15 or later) before installation;** running the installer against an unsupported Node.js version can leave the shell partially configured.

Choosing between Outside-In and Inside-Out

Outside-In and Inside-Out integration are not mutually exclusive, and some teams may reasonably use both, depending on the developer and the project. The table below summarizes the practical trade-offs to weigh when deciding which pattern to standardize on for a given team.

Table: Outside-In versus Inside-Out: practical trade-offs.

Consideration	Outside-In	Inside-Out
Where IBM Bob runs	Locally, on the developer's machine	Inside the BAS dev space itself
Setup effort	Higher — requires building and registering a custom MCP server per developer machine	Lower — a single install command inside the dev space, plus an API key
Access to IBM Bob's local capabilities	Full — local modes, rules, memory, and other MCP servers remain available regardless of BTP tenant	Limited to what is explicitly copied into the dev space (skills, rules, modes)
Credential exposure surface	BTP credentials or OAuth token stored on the local machine, in mcp.json	IBM Bob API key stored session-scoped or persisted inside the dev space
Best fit	Individual developers with a persistent local environment and their own BAS trial or dev tenant	Fast onboarding, shared or ephemeral dev spaces, and developers who primarily work inside BAS
Dependency on a custom, unofficial component	Yes — bas-mcp-server is a custom implementation (see the Roadmap Considerations section)	Yes — the Bob shell extension inside BAS is likewise not an SAP-released artifact

Project managers scoping a rollout should treat this as a per-team decision rather than a single organization-wide mandate, and architects should apply the same SAP-released-tooling-first governance principle described elsewhere in this publication: if SAP releases an official, equivalent capability for either pattern, it should be evaluated for adoption ahead of the custom components described in this section.

Roadmap considerations

The `bas-mcp-server` described in the Category 1: Outside-In Integration section is a custom implementation. It wraps the BAS File API that SAP exposes for dev space management, and it is stable in that sense — the underlying API has been available since BAS's initial release. What it does not do is use an SAP-released MCP artifact; it was built to fill a gap that SAP's current official MCP catalog does not cover. The Bob shell extension used in the Inside-Out pattern (see Category 2: Inside-Out Integration) is similarly a purpose-built component rather than an SAP-released one.

Two developments are worth watching. First, SAP has committed to expanding its official MCP surface across BTP product lines. If SAP releases an official BAS MCP server — covering the dev space lifecycle, extension installation, or more — the custom server described in the Category 1: Outside-In Integration section should be retired in favor of it, following the SAP-released-tooling-first principle established elsewhere in this publication. Second, SAP Business Application Studio is a hosted environment, but SAP has also offered a VS Code extension (SAP Business Application Studio for VS Code) that runs the same tooling locally. As that extension matures, the integration pattern may shift from the remote File API model to a local file-system model, simplifying the MCP bridge considerably.

Neither development changes the fundamental architecture described in this section. Outside-in integration — local AI agent, remote development environment, MCP bridge — and inside-out integration — AI agent embedded directly in the remote environment — are both durable patterns regardless of which specific tools sit at each end.

Summary and key takeaways

SAP BTP UI5 development built on CAP, Node.js, and JavaScript is typically carried out in SAP Business Application Studio, a cloud-hosted development environment organized around per-developer dev spaces. IBM Bob supports this workflow across requirement gathering, code generation, review, reverse engineering, and deployment, through one of two integration patterns.

The Outside-In pattern puts IBM Bob in control of a SAP BAS dev space from the developer's local machine. A single Node.js MCP server — `bas-mcp-server` — provides the bridge, authenticating to BAS through either a password-grant

flow or a pre-supplied OAuth 2.0 access token, and exposing seven tools that cover the full development lifecycle: reading, writing, searching, folder management, and command execution. UI5 projects arrive in the local Bob workspace through standard Git operations and become the subject of AI-assisted conversations that generate controllers and views, refactor OData service calls, produce technical specifications, write QUnit test suites, and trace complex data flows across a codebase.

The Inside-Out pattern instead brings IBM Bob to the code, installing a Bob shell directly inside the BAS terminal and authenticating it with an API key generated from the developer's IBM Bob account. This lowers setup effort and suits shared or ephemeral dev spaces, at the cost of leaving IBM Bob's local-only capabilities behind and introducing a credential — the API key — that must be managed carefully if persisted inside the dev space.

Four principles govern productive and secure use of either pattern:

- **Scope tasks to match a single focused session.** Broad, open-ended refactoring prompts produce diffuse results; specific, bounded tasks produce artifacts the developer can review and commit immediately.
- **Review before every write or execute confirmation.** The tool confirmation prompt is not overhead — it is the review gate that separates AI assistance from autonomous modification of shared infrastructure.
- **Manage credentials — BTP passwords, OAuth tokens, and IBM Bob API keys alike — with the same discipline applied to any other privileged credential**, including rotation, scoping, and avoiding persistence in shared environments.
- **Prefer SAP-released tooling wherever it exists, and monitor SAP's expanding MCP catalog.** Both the `bas-mcp-server` and the Bob shell extension fill a current gap; an official replacement should be adopted when one becomes available.

> **Note:** This section documents a custom MCP server implementation (`bas-mcp-server`) built against the SAP BAS File API, and a custom Bob shell extension for the Inside-Out pattern, both as they existed at the time of writing. Developers should verify the current BAS API endpoint structure and authentication behavior against the official SAP BAS documentation before deploying either pattern in a production environment. The `bas-mcp-server` source code is located at `Workspace/bas-mcp-server/src/index.ts` in the IBM Bob workspace described throughout this publication.

IBM Bob in SAP Integration: Governed AI assistance for SAP Cloud Integration

This section describes the application of IBM Bob to SAP Cloud Integration (SAP Integration Suite, commonly referred to as CPI) development and migration engagements. It covers the full project lifecycle — from requirement analysis through Groovy script generation, iFlow design, PI/PO migration, runtime monitoring, and deployment — and walks through ci-mcp-server, a custom, project-built Model Context Protocol (MCP) server created specifically for this publication to demonstrate how IBM Bob can be connected to a live SAP BTP Integration Suite tenant. This section places particular emphasis on the governance architecture that separates AI-generated recommendations from privileged SAP change and deployment actions, and is written for a mixed audience of SAP integration developers, solution architects, project managers, and senior management.

> **Important:** ci-mcp-server is a demonstration artifact, not an SAP or third-party product.
>
> ci-mcp-server is not released, distributed, or supported by SAP, IBM, or any other vendor. It is a custom MCP server that this section's authors built for the sole purpose of illustrating what an IBM Bob – SAP Integration Suite connection can look like and what it can enable. Every code sample, tool definition, and configuration file in this section describing ci-mcp-server is an example to be built and maintained by the adopting project team — there is no package to download and no vendor to raise a support ticket

> with. Teams that adopt this pattern take on full ownership of the server's implementation, security review, and maintenance, and should follow the SAP-released-tooling-first principle described elsewhere in this publication: if SAP or another trusted vendor later publishes an official MCP server for Integration Suite, it should be evaluated ahead of continued investment in a custom implementation.

SAP Cloud Integration development involves a recurring set of tasks — designing integration flows, writing Groovy scripts, diagnosing message failures, validating security material, and migrating legacy PI/PO objects — each of which benefits from AI assistance grounded in the actual state of a live Integration Suite tenant. This section documents how IBM Bob is applied to that work, and, just as importantly, how the integration is governed so that AI-generated recommendations remain clearly separated from the privileged actions that change a production SAP landscape.

The central technical enabler discussed in this section is ci-mcp-server. To be clear about what this is: ci-mcp-server is not a standard MCP server released or supported by SAP, IBM, or any other party. It is a custom, purpose-built server that must be created by the adopting project team, developed here specifically to demonstrate the connection capability between IBM Bob and SAP BTP Integration Suite and to give the rest of this section a concrete, working example to reason about. It exposes the SAP Cloud Integration OData API as a set of structured tools IBM Bob can call. The central governance principle, examined in the Governed Architecture Overview section, is that read-only observation of a tenant is treated differently from any action that mutates it: the former is available to IBM Bob by default, while the latter passes through an explicit human approval gate before execution.

Scope

This section covers the architecture and governance model for connecting IBM Bob to a SAP BTP Integration Suite tenant; the ci-mcp-server tool inventory, authentication, and setup procedure; IBM Bob's capabilities across the CPI development and PI/PO migration lifecycle; runtime monitoring and security material validation; and the reusable Groovy script library that results from this workflow. It assumes familiarity with SAP Integration Suite concepts (integration flows, adapters, Message Monitor, Security Material) and with the general MCP configuration model introduced elsewhere in this publication.

Governed architecture overview

IBM Bob interacts with SAP Cloud Integration through a layered architecture in which every capability is assigned to one of two paths: an observation path, available by default, and a privileged mutation path, gated behind explicit human approval. The following figure presents the complete reference architecture, spanning the IBM Bob boundary, the SAP BTP trust boundary, and the operational evidence layer that provides audit and assurance for both.

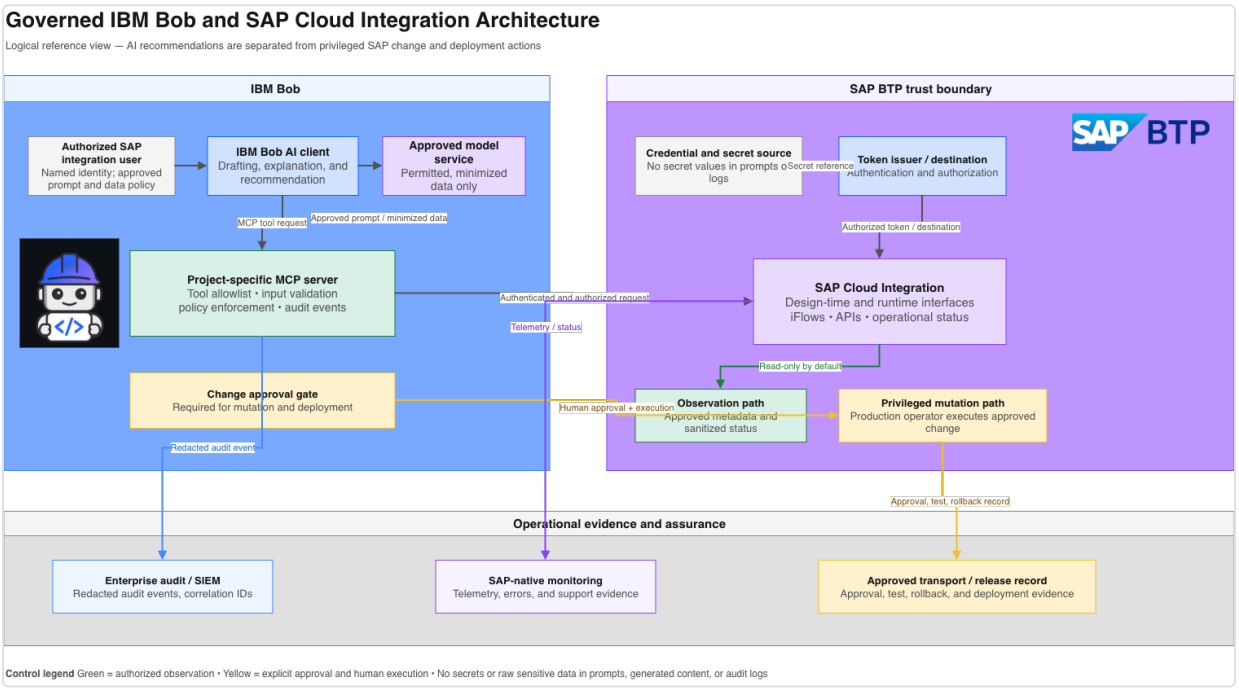


Figure: Governed IBM Bob and SAP Cloud Integration reference architecture

Control legend:

- Green denotes authorized observation.
- Amber denotes an action that requires explicit approval and human execution.
- No secrets or raw sensitive data may appear in prompts, generated content, or audit logs anywhere in this architecture — this is a hard constraint on the design, not a configuration option.

The IBM Bob boundary

On the left of the architecture diagram above, a named, authorized SAP integration user operates the IBM Bob AI client, which drafts, explains, and recommends against an approved model service configured to receive only minimized, permitted data. The client issues MCP tool requests to a project-specific MCP server — ci-mcp-server — which enforces a tool allowlist, validates inputs, applies policy, and emits audit events for every call it handles. A change approval gate sits alongside this server: any action classified as a mutation is routed through this gate and requires explicit human approval before it executes.

The SAP BTP trust boundary

On the right, the SAP BTP environment holds the credential and secret source and the token issuer that authenticates and authorizes every request arriving from the MCP server — no secret values ever appear in a prompt or in a log. Authenticated, authorized requests reach SAP Cloud Integration's design-time and runtime interfaces along two distinct paths:

- The **observation path** (shown in green): read-only by default. It returns approved metadata and sanitized status information, such as an iFlow definition, a message-monitor trace, or a runtime status, without requiring per-call approval.
- The **privileged mutation path** (shown in amber): used for any action that changes tenant state, such as deploying an integration flow or creating a new credential alias. This path only executes once the change approval

gate has recorded human approval, and execution itself is carried out by an authorized production operator rather than autonomously by the AI client.

Operational evidence and assurance

Every interaction leaves a trace appropriate to its sensitivity. Redacted audit events from the MCP server flow to enterprise audit and SIEM tooling, with correlation IDs but no sensitive payload content. Telemetry and status information from the observation path flows to SAP-native monitoring. Every approval, test, and rollback decision taken at the change approval gate is captured in an approved transport or release record. Together, these three evidence streams let an auditor reconstruct what IBM Bob observed, what it recommended, and — separately — who approved and executed any resulting change.

Table: Governed architecture component summary.

Component	Boundary	Role
IBM Bob AI client	IBM Bob	Drafts, explains, and recommends; issues MCP tool requests; never executes a privileged change directly.
Approved model service	IBM Bob	Receives only permitted, minimized data as part of the model request.
Project-specific MCP server (ci-mcp-server)	IBM Bob	Enforces the tool allowlist, validates inputs, applies policy, and emits audit events for every call.
Change approval gate	IBM Bob	Required checkpoint for any mutation or deployment action before it reaches the privileged path.
Credential and secret source	SAP BTP	Holds live credentials; no secret values appear in prompts or logs.
Token issuer / destination	SAP BTP	Authenticates and authorizes every request before it reaches Integration Suite.
SAP Cloud Integration	SAP BTP	Hosts design-time and runtime interfaces: iFlows, APIs, and operational status.
Observation path	SAP BTP	Read-only by default; returns approved metadata and sanitized status.
Privileged mutation path	SAP BTP	Production operator executes an approved change once the gate has recorded approval.
Enterprise audit / SIEM	Operational Evidence	Receives redacted audit events and correlation IDs for every MCP tool call.
SAP-native monitoring	Operational Evidence	Receives telemetry, errors, and support evidence from the observation path.
Approved transport / release record	Operational Evidence	Captures approval, test, and rollback evidence for every executed change.

This architecture eliminates the need for a developer to context-switch between the IDE and the SAP BTP Cockpit for the observation activities that make up the majority of day-to-day integration work — reading flow definitions,

checking message traces, verifying runtime status — while preserving a conventional, auditable change-management gate for anything that mutates the tenant.

The ci-mcp-server

> **Important: This is an example implementation, created for this Redbooks publication, it has not been supplied by SAP**
>
> ci-mcp-server is a custom MCP server that this section's authors designed and built expressly to demonstrate how IBM Bob can connect to SAP BTP Integration Suite. It is not a product, sample repository, or supported artifact published by SAP, IBM, or any other vendor. Everything in this section — the registration entry, the tool inventory, and the authentication setup — describes a server that a project team builds and owns for itself, using the SAP Cloud Integration OData v2 API as its integration point. Treat the code and configuration shown here as a reference design to adapt, review, and secure for your own environment, not as a package to install.

The ci-mcp-server is a project-specific MCP server that wraps the SAP Cloud Integration OData v2 API. It is registered in the project's central mcp.json configuration file and distributed to developers as part of standard project setup.

Registration

The following code block provides context to understand the sample mcp server registration steps.

```
{

"mcpServers": {

"ci-mcp-server": {

"command": "npm",

"args": ["start"],

"cwd": "/workspace/ci-mcp-server",

"env": { "NODE_ENV": "development" },

"metadata": {

"description": "SAP Cloud Integration MCP Server — manages integration flows, monitors messages, and handl

"configurationNotes": "Requires SAP BTP credentials. Provide XSUAA, Destination, and Connectivity service

             }
           }
         }
}
```

Exposed tools and governance path

Each tool the server exposes is classified against the observation/mutation model introduced in the Governed Architecture Overview section. Observation tools are available to IBM Bob without per-call approval; mutation tools require the change approval gate.

Table: ci-mcp-server tool inventory, classified by governance path.

Tool	Function	Governance path
list_integration_flows	Enumerate all integration flows within a package	Observation
get_iflow_definition	Retrieve the complete BPMN/XML definition of an integration flow	Observation
monitor_messages	Query the Message Monitor for processing status and error traces	Observation
manage_security_credentials	List existing credential aliases (observation); create or update an alias (mutation)	Mixed — see note
deploy_iflow	Trigger deployment of an integration flow to the runtime tenant	Mutation — approval gate
get_runtime_status	Retrieve the operational status of a deployed flow	Observation

For `manage_security_credentials`, listing existing aliases is a read-only lookup and follows the observation path. Creating or updating an alias changes tenant state and therefore follows the privileged mutation path: IBM Bob may propose the change and generate the corresponding API call, but a production operator completes it after approval, consistent with the pattern described in the Runtime monitoring and diagnostics section.

Authentication

The server authenticates to SAP BTP using Cloud Foundry service bindings provided in a local `default-env.json` file. This file contains XSUAA, Destination Service, and Connectivity Service credentials. It is excluded from version control by `.gitignore` and is created locally by each developer from the project-supplied template.

```
{
"VCAP_SERVICES": {
"xsuaa": [{
"credentials": {
"clientid":     "${XSUAA_CLIENT_ID}",
"clientsecret": "${XSUAA_CLIENT_SECRET}",
"url":          "https://<subaccount>.authentication.eu10.hana.ondemand.com"
  }
}],

"destination": [{
"credentials": {
"uri":          "https://destination-configuration.cfapps.eu10.hana.ondemand.com",
"clientid":     "${DEST_CLIENT_ID}",
"clientsecret": "${DEST_CLIENT_SECRET}"

  }
}]
}
}
```

> **Note:** Credential files must never be committed to any repository. All sensitive values are supplied through environment-specific templates, following the governance model (see Working with IBM Bob, under Centralized IBM Bob configuration and distribution for SAP teams).

IBM Bob capabilities across the CPI development lifecycle

Requirement gathering and technical analysis

IBM Bob can process a functional specification document and extract structured integration requirements. Given an FS document, Bob identifies source and target systems, communication protocols (REST, SOAP, SFTP, IDoc), security requirements (OAuth 2.0, Basic Authentication, mTLS), message formats, transformation logic, and error-handling constraints. The output is a technical design checklist aligned to CPI iFlow components, which serves as the basis for the development task list.

Integration flow design

IBM Bob assists in the design of integration flow structures through the following activities:

- Recommending adapter types appropriate to the identified protocols

- Defining the sequence of processing steps (Content Modifier, Groovy Script, Router, Splitter, Aggregator)
- Identifying message mapping requirements and transformation patterns
- Specifying security material references and credential alias naming conventions

Where ci-mcp-server is connected, Bob reads existing iFlow definitions from the tenant — an observation-path call — before making recommendations, ensuring that generated designs are consistent with the established integration patterns in the project.

Groovy script generation

Groovy scripting is the primary customization mechanism within SAP Integration Suite. IBM Bob generates production-ready Groovy scripts that conform to CPI best practices, enforced through the project's CPI Mode rules (see CPI mode definition). The following standards are applied to every generated script:

- The `messageLog` API is used for all diagnostic output; `System.out.println` is prohibited.
- Typed body access using `getBody(String.class)`; raw InputStream access is avoided.
- Explicit try-catch blocks on all external calls.
- Connection and read timeouts on all `HttpURLConnection` instances.
- Message headers are used for routing decisions; message properties for cross-step state persistence.

Example — JSON Transformation (generated by IBM Bob)

```
import com.sap.gateway.ip.core.customdev.util.Message
import groovy.json.JsonSlurper
import groovy.json.JsonBuilder
def Message processData(Message *message*) {
def messageLog = messageLogFactory.getMessageLog(message)
try {
def data = new JsonSlurper().parseText(message.getBody(String.class))
def result = [
processedOrders: data.orders?.collect { *order* ->

            [ orderId:      order.id,

customerName: order.customer,
grandTotal:   order.amount * 1.18,
status:       "processed",
processedDate: new Date().format("yyyy-MM-dd'T'HH:mm:ss'Z'") ]

        },

summary: [ totalOrders: data.orders?.size(), processedBy: "CPI_Integration" ]

    ]
    message.setBody(new JsonBuilder(result).toPrettyString())
    message.setHeader("RecordCount", result.processedOrders?.size()?.toString() ?: "0")
    messageLog.setProperty("TransformationStatus", "Success")
} catch (Exception e) {
    messageLog.setProperty("Error", e.getMessage())
    message.setBody(new JsonBuilder(
        [error: true, message: e.getMessage(), timestamp: new Date().toString()]
    ).toString())
}

return message
}
```

```
*// Made with IBM Bob*
```

Example — Content-based routing (generated by IBM Bob)

The following script sets message headers that are consumed by a downstream CPI Router step. Bob generates both the routing logic and the header names referenced in the Router step conditions, ensuring consistency between the Groovy script and the iFlow configuration.

```
import com.sap.gateway.ip.core.customdev.util.Message

import groovy.json.JsonSlurper

def Message processData(Message *message*) {

def messageLog = messageLogFactory.getMessageLog(message)

try {

def data   = new JsonSlurper().parseText(message.getBody(String.class))

def amount = data.amount as Double

def type   = data.customerType ?: "STANDARD"

def destination = amount > 10000 ? "HighValueOrderReceiver" :

                      amount > 5000  ? "MediumValueOrderReceiver" : "StandardOrderReceiver"

if (type == "PREMIUM") destination = "PremiumCustomerReceiver"

    message.setHeader("RoutingDestination", destination)

    message.setHeader("RouteToHighValue",   (amount > 10000).toString())

    message.setHeader("RouteToPremium",     (type == "PREMIUM").toString())

    messageLog.setProperty("RoutingDecision", destination)

} catch (Exception e) {

    message.setHeader("RoutingDestination", "ErrorHandlingReceiver")

    message.setProperty("ErrorOccurred", "true")

}

return message

}

*// Made with IBM Bob*
```

Error handling and retry patterns

IBM Bob generates structured error handling logic incorporating exponential backoff retry and dead-letter queue routing. The retry count is persisted as a message property across iFlow iterations; the backoff interval is calculated

as a power function of the attempt number. On exhaustion of the maximum retry threshold, the message is routed to a configurable dead-letter channel and an error notification header is set for downstream alerting.

External system enrichment

IBM Bob generates HTTP client code for message enrichment patterns, including typed connection handling, OAuth Bearer token injection, configurable timeouts, and graceful fallback — ensuring that an enrichment failure does not abort the primary integration flow.

XML processing

For XML payloads, IBM Bob generates scripts using Groovy's `XmlSlurper` for parsing and `MarkupBuilder` for output construction. Generated scripts navigate XML structures using path expressions, apply transformations, aggregate summary data, and produce well-formed output documents.

IBM Bob for CPI migration projects

PI/PO to SAP Integration Suite migration engagements involve the systematic conversion of existing integration objects — flows, message mappings, Java user-defined functions, and connectivity configurations. IBM Bob supports each phase of this process, within the same observation/mutation governance model described in the SAP ABAP section.

Table: IBM Bob capability by migration phase.

Migration phase	IBM Bob capability
Assessment	list_integration_flows enumerates all objects; Bob analyzes definitions for custom code, adapter types, and certificate dependencies.
Design	Bob reads existing iFlow XML and generates the structural equivalent for SAP Integration Suite.
Development	Bob converts Java message mappings and UDFs to Groovy scripts; reconfigures adapter parameters; creates security credential references.
Testing	Bob generates test payloads from message schemas; interprets Message Monitor error traces; proposes mapping corrections.
Deployment	deploy_iflow triggers tenant deployment once approved at the change approval gate; get_runtime_status and monitor_messages verify successful execution.

A representative migration task — converting a PI/PO Java mapping to a CPI Groovy script — is accomplished by providing Bob with the source Java code and requesting a conversion that applies `JsonBuilder`, `messageLog`, structured exception handling, and correct header assignments. Bob performs this conversion in a single interaction, producing a deployable script that conforms to the project's coding standards and is ready for developer review ahead of deployment.

Runtime monitoring and diagnostics

With ci-mcp-server connected, IBM Bob can diagnose integration failures without the developer leaving the IDE. When a flow exhibits processing errors, Bob invokes `monitor_messages` to retrieve error traces — an observation-path call — correlates them against the iFlow definition obtained via `get_iflow_definition`, and identifies the root cause, typically a null reference, an incorrect XPath expression, or a malformed payload. Bob then proposes a corrected script that addresses the identified defect, which the developer reviews and deploys through the normal change process.

Security material validation

A frequent cause of deployment failure is the absence of credential aliases in Security Material. IBM Bob performs pre-deployment validation by reading all credential references from the iFlow definition and verifying their existence in the BTP Security Material store — both read-only, observation-path operations. Any missing alias is reported with the recommended resolution.

> **Important:** Where creation of a missing alias is authorized, Bob generates the corresponding API call with placeholder values, but the call is a mutation and is executed only after passing the change approval gate described in the Governed architecture overview section — never autonomously.

The CPI Groovy script library

The project workspace includes a library of reusable CPI Groovy script patterns under `CPI_Groovy_Scripts/`, generated using IBM Bob. This library serves as a reference and starting point for integration developers.

Table: CPI Groovy script library — script patterns

Script	Pattern
01_basic_message_processing.groovy	Body access, header and property management, audit logging
02_json_transformation.groovy	JSON parsing and structural transformation using JsonSlurper/JsonBuilder
03_xml_processing.groovy	XML parsing and output generation using XmlSlurper/ MarkupBuilder
04_dynamic_routing.groovy	Content-based routing header injection for CPI Router step conditions
05_external_api_call.groovy	HTTP enrichment with timeout configuration and graceful fallback
06_error_handling_retry.groovy	Exponential backoff retry logic with dead-letter queue routing
07_message_splitting_aggregation.groovy	Batch message splitting with correlation ID and aggregation logic

All scripts in the library are traceable to IBM Bob through the `// Made with IBM Bob` comment at the end of each file.

Configuration and setup

Prerequisites

Verify the following prerequisites:

- Access to a SAP BTP subaccount with Integration Suite provisioned
- XSUAA, Destination Service, and Connectivity Service instances with bound credentials
- Node.js runtime for ci-mcp-server
- IBM Bob IDE with MCP server support enabled

Initial setup procedure

1. Clone the project's central Bob configuration repository and execute the `setup.sh` bootstrap script to populate the `.bob/` workspace directory.
2. Copy `templates/default-env.json.template` to the ci-mcp-server directory and populate all credential placeholders with values obtained from the SAP BTP Service Key (see Creating the required BTP service instances and destination).
3. Start ci-mcp-server using `npm start` from its working directory.
4. Enable the ci-mcp-server entry in `mcp.json` by setting `"disabled": false`.

Creating the required BTP service instances and destination

ci-mcp-server depends on three SAP BTP service instances and one destination, all created in the BTP subaccount where Integration Suite is provisioned. The credentials obtained from each service instance are placed into the local `default-env.json` file; they are never committed to version control.

```
BTP Subaccount (Integration Suite)
├── Service Instance: Authorization & Trust Management (XSUAA)  → fills xsuaa block
├── Service Instance: Destination Service                         → fills destination block
├── Service Instance: Connectivity Service (optional)           → fills connectivity block
└── Destination: CPI_DESTINATION                                → points to CPI runtime URL
```

XSUAA service instance

The XSUAA instance provides the OAuth 2.0 client credentials the server uses to authenticate all API calls to the Integration Suite tenant.

First, in BTP Cockpit, navigate to the subaccount → Services → Service Marketplace.

Then, search for "Authorization and Trust Management Service" and select Create.

Table: Authorization and Trust Management service instance field values

Field	Value
Service	Authorization and Trust Management Service
Plan	application
Instance name	ci-mcp-server-xsuaa

In the JSON parameters field, paste the following:

```
{

"xsappname": "ci-mcp-server",

"tenant-mode": "dedicated",

"scopes": [
{ "name": "$XSAPPNAME.read", "description": "Read CPI artifacts" }


],

"role-templates": [
{ "name": "Viewer", "scope-references": ["$XSAPPNAME.read"] }

]
}
```

Next, select Create. Once the instance is ready, open it and select Service Keys → Create Service Key. Name it `ci-mcp-server-key`.

Finally, open the key and copy the following values into `default-env.json` under the `xsuaa` block:

```
"xsuaa": [{

"label": "xsuaa",

"credentials": {

"clientid":     "<clientid from service key>",

"clientsecret": "<clientsecret from service key>",

"url":          "<url from service key>",

"xsappname":    "ci-mcp-server"

}

}]
```

Destination service instance

The Destination Service instance allows the server to resolve the logical destination name `CPI_DESTINATION` to the actual CPI runtime URL at runtime.

First select, Services → Service Marketplace → search "Destination" → select Create.

Table: Cloud Foundry Destination service instance field values

Field	Value
Service	Destination
Plan	lite
Instance name	ci-mcp-server-destination

No JSON parameters are required. Next, select 'Create'.

Then, open the instance → Service Keys → Create Service Key → name it `ci-dest-key`.

Finally, copy the following values into `default-env.json` under the `destination` block:

```
"destination": [{
"label": "destination",
"credentials": {
"clientid":     "<clientid from service key>",
"clientsecret": "<clientsecret from service key>",
"uri":          "<uri from service key>",
"url":          "<url from service key>"
}
}]
```

Creating the CPI_DESTINATION

The `CPI_DESTINATION` entry is the destination the server uses to call the Integration Suite OData APIs. It must be named exactly `CPI_DESTINATION` to match the value in `ci-api-config.json`.

First, In BTP Cockpit → the subaccount → Connectivity → Destinations → New Destination.

Table: Destination service instance configuration field values

Field	Value
Name	CPI_DESTINATION
Type	HTTP
URL	https://.it-cpi018-rt.cfapps..hana.ondemand.com
Proxy Type	Internet
Authentication	OAuth2ClientCredentials
Client ID	clientid from the Process Integration Runtime service key
Client Secret	clientsecret from the Process Integration Runtime service key
Token Service URL	https://.authentication..hana.ondemand.com/oauth/token

Then, select Save, then Check Connection. A 200 OK response confirms the destination is correctly configured.

> **Note:**
>
> **Finding the CPI runtime URL** -
>
> In BTP Cockpit → the subaccount → Instances and Subscriptions → locate Integration Suite — the runtime URL appears in the subscription details or in the Process Integration Runtime service instance (plan: api) service key, under `url`.
>
> **Process Integration Runtime credentials** -
>
> Navigate to the subaccount → Instances and Subscriptions → find the Process Integration Runtime service instance with plan `api` → open its service key — the `clientid`, `clientsecret`, and `tokenurl` values are what the destination needs.

Connectivity service instance (optional)

The Connectivity Service instance is only required if integration flows connect to on-premise SAP systems via the SAP Cloud Connector. For cloud-to-cloud Integration Suite tenants, the `connectivity` block can be removed from `default-env.json`.

If on-premise connectivity is required, complete the following steps:

1. Services → Service Marketplace → search "Connectivity" → Create with plan `lite`.
2. Create a service key and copy `clientid`, `clientsecret`, and `onpremise_proxy_host` into the `connectivity` block of `default-env.json`.

Completed default-env.json

With all three service keys populated, the completed `default-env.json` has the following structure:

```
{
"VCAP_SERVICES": {
"xsuaa": [{
"label": "xsuaa",
"credentials": {
"clientid":     "<xsuaa clientid>",
"clientsecret": "<xsuaa clientsecret>",
"url":          "https://<subdomain>.authentication.<region>.hana.ondemand.com",
"xsappname":    "ci-mcp-server"

  }
}],

"destination": [{
"label": "destination",
"credentials": {
"clientid":     "<destination clientid>",
"clientsecret": "<destination clientsecret>",
"uri":          "https://destination-configuration.cfapps.<region>.hana.ondemand.com",
"url":          "https://<subdomain>.authentication.<region>.hana.ondemand.com"

  }
}]

}
}
```

> **Important:** `default-env.json` must be listed in `.gitignore` and must never be committed to any repository. It contains live BTP service credentials.

CPI mode definition

A dedicated CPI Mode is defined in `custom_modes.yaml` to scope IBM Bob's behavior for integration development work. The mode configures ci-mcp-server as the required MCP connection, references applicable skills and rules, enforces the Groovy coding standards described in the Groovy Script Generation section, and encodes the governance boundary from the Governed Architecture Overview section directly into IBM Bob's operating instructions.

```
customModes:

- slug: cpi-mode

name: ☁️ CPI Integration Mode

roleDefinition: >

  You are an expert SAP Cloud Integration developer.

  Always read the live iFlow definition via ci-mcp-server before

  generating or modifying any Groovy script or adapter configuration.

 Apply the following standards to all generated Groovy:

  - Use messageLog exclusively; never use System.out.println

  - Use getBody(String.class) for typed body access

  - Wrap all external calls in try-catch with connection and read timeouts

  - Use message headers for routing; use message properties for cross-step state

customInstructions: >
  Validate iFlow context before code generation.

  Always include structured error responses in catch blocks.

  Never generate code without consulting the live iFlow definition first.

  Treat deploy_iflow and any create/update call within manage_security_credentials as proposals only:
  draft the change and explain its effect, but do not represent it as executed until the developer has app

groups:
  - read

  - edit

  - execute

  - mcp
```

The `customInstructions` block above is the point at which the governance model in the Governed Architecture Overview section becomes an operating rule rather than an architectural aspiration. Project teams adopting this pattern should treat this instruction as mandatory, not optional boilerplate: it is what keeps IBM Bob's language consistent with its actual authority — proposing a change is not the same as making one.

Value summary

The following table summarizes the productivity impact of IBM Bob across representative CPI development activities. Estimates are based on project experience and are provided as indicative guidance rather than a guaranteed outcome for any specific engagement.

Table: Indicative productivity impact by activity.

Activity	Conventional approach	With IBM Bob
Groovy script development	2–4 hours per script	Significantly reduced through AI-assisted generation, subject to developer review
Failure diagnosis	30–60 minutes per incident (manual log inspection)	Minutes — Bob reads the Message Monitor and cross-references the iFlow
Java mapping conversion (PI/PO migration)	4–8 hours per mapping	Single AI-assisted interaction, followed by developer review and adjustment
Flow inventory and assessment	1–2 days for large landscapes	Automated via list_integration_flows
Pre-deployment credential validation	Manual cross-reference	Automated via manage_security_credentials (observation path)
Integration flow documentation	Separate effort, often deferred	Generated on demand from the live iFlow definition

IBM Bob provides structured, governed AI assistance across the full SAP Integration Suite development and migration lifecycle. ci-mcp-server is the technical enabler that gives IBM Bob direct access to a live SAP BTP tenant, allowing it to read, analyze, generate, and validate integration artifacts within a single development session. The governance architecture described in the Governed architecture overview section — separating the observation path from the privileged mutation path, and gating every change behind human approval — is what makes this access safe to extend to a live tenant in the first place.

The CPI Mode configuration ensures that all generated code conforms to project coding standards and CPI best practices, and that IBM Bob's own language about what it has done stays honest: a drafted deployment is a proposal, not an executed change, until a developer has approved it.

For migration engagements, IBM Bob reduces the manual effort associated with flow inventory, mapping conversion, and regression testing — activities that represent a significant proportion of total migration cost in large PI/PO landscapes. For greenfield development, it accelerates script authoring, debugging, and documentation while enforcing consistency across all project contributors, and its audit trail gives project managers and senior management a verifiable record of what was recommended, what was approved, and what was ultimately deployed.

> **Note:** ci-mcp-server, as used throughout this section, is a custom example server built by this publication's authors to demonstrate the IBM Bob – SAP Integration Suite connection — it is not an SAP-released, IBM-released, or third-party-supported product. A project adopting this pattern is responsible for building, securing, and maintaining its own equivalent server, using the SAP Cloud Integration OData v2 API as it existed at the time of writing. Teams should verify current API behavior against the official SAP Integration Suite documentation, follow the SAP-released-tooling-first principle described elsewhere in this publication should an official equivalent later become available, and treat the observation/mutation governance model in the *Governed Architecture Overview* section as a minimum baseline to be strengthened, not weakened, to fit their organization's change-management policy.

SAP documentation with Bob

SAP documentation challenges are well known in the industry: inconsistent quality across consultants, knowledge locked in individuals, templates that vary by project, and time pressure that leads to incomplete artifacts. IBM Bob addresses these challenges in several ways:

- **Consistency:** Bob follows custom rules and templates, ensuring every document produced adheres to the same structure and quality bar, regardless of who runs the prompt.
- **Speed:** A Functional Specification that takes a consultant half a day to write can be drafted by Bob in minutes, freeing the consultant to review and validate rather than author from scratch.
- **Traceability:** Bob's agentic mode can read ABAP source files, WRICEF lists, requirement workbooks, and transport logs to ground document content in actual system artifacts.
- **Enterprise Security:** IBM Bob keeps all data within IBM's enterprise boundary. No source code, SAP configuration, or customer data is sent to external third-party AI services.

IBM Bob architecture and core capabilities

Understanding how IBM Bob works is essential to designing effective documentation workflows.

Modes

IBM Bob operates through a set of built-in and customizable modes. Each mode defines Bob's role, permitted tools, and behavioral instructions. The key modes relevant to SAP documentation are:

- **Agent Mode:** The primary agentic mode. Bob reads files, reasons across the full context, creates and modifies documents autonomously, and can run terminal commands. This is the mode used for the bulk of document generation work.
- **Plan Mode:** Used to design, strategize, and outline before authoring. Ideal for structuring a complex document like a Business Blueprint or Solution Architecture before generating the full content.
- **Ask Mode:** Read-only conversational mode. Use this to query Bob about SAP concepts, explain existing ABAP code, or clarify requirements — without making any file changes.
- **Custom Modes:** Teams can define SAP-specific modes, such as an 'SAP Functional Consultant' mode with pre-loaded SAP Activate methodology instructions, standard WRICEF templates, and naming conventions — ensuring consistent output across the project team.

Context mentions

IBM Bob allows users to inject specific content into the chat context using @ mentions. This is a key mechanism for SAP documentation work:

- @/path/to/file.abap: Includes the contents of an ABAP source file so Bob can analyze it and generate a Technical Specification from the actual code.
- @/path/to/requirements.xlsx: Attaches an Excel requirements workbook so Bob can extract user stories and acceptance criteria for a Functional Specification.
- @/docs/template.docx: Provides a Word template so Bob generates output that matches the corporate document standard.
- @terminal: Includes recent terminal output, useful for injecting SAP transport logs, ABAP syntax check results, or test run outputs.

Rules and custom instructions

Rules are persistent, project-level instructions that Bob follows in every conversation. For SAP projects, teams configure rules to enforce: document naming conventions (for example, FS-<ModuleName>-<ObjectName>-

v1.0.docx), standard SAP Activate phase gating, required sections for each document type, version and author metadata, and language and formality standards for client-facing documents.

Agentic file creation

In Agent mode, IBM Bob can autonomously create fully-structured Word (.docx), Excel (.xlsx), Markdown (.md), and plain-text files. It does not merely output text in the chat — it writes the actual document file to the project workspace. This means the output is immediately usable, version-controllable, and shareable with the project team.

IBM Bob in the SAP project lifecycle

SAP Activate — IBM's preferred methodology for SAP S/4HANA implementations — consists of six phases: Discover, Prepare, Explore, Realize, Deploy, and Run. IBM Bob contributes to documentation production across every phase.

Discover

During Discover, Bob can generate the following assets:

- Business Case summaries from executive requirement inputs
- Fit-Gap Assessment templates pre-populated with standard SAP S/4HANA scope
- Project Charter drafts based on engagement scope information

Prepare

In Prepare, Bob assists with the following assets:

- Project Plan documentation
- System Landscape documents
- Team Role and Responsibility matrices (RACI)
- Governance Framework documents

Explore

Explore is where the highest volume of Functional Specifications is produced. Bob generates the following assets:

- Business Process Procedures (BPP)
- Fit-Gap Analysis reports
- WRICEF (Workbooks, Reports, Interfaces, Conversions, Enhancements, Forms) lists
- Configuration Workbooks for each SAP module

Realize

In Realize, Technical Specifications and test scripts dominate. Bob generates the following assets:

- ABAP Object Technical Specifications (from source code)
- Interface Technical Design Documents
- Data Migration Strategy and Mapping documents
- Unit Test Scripts, and Integration Test Plans

Deploy and Run

In Deploy and Run, Bob supports the production of the following assets:

- Cutover Plans
- Training Materials and User Guides
- Knowledge Transfer documents
- Post Go-Live Support runbooks

Generating functional specifications with IBM Bob

A Functional Specification (FS) in SAP describes what a business process or custom development object must do, from the business perspective, without specifying how the technical implementation is achieved. IBM Bob can draft, structure, and populate Functional Specifications using several complementary approaches.

Standard sections of an SAP functional specification

A well-formed SAP Functional Specification typically contains: Document Header (Title, Author, Version, Date, Approver), Business Context and Objective, Scope and Out-of-Scope, Business Process Overview, Detailed Functional Requirements (including AS-IS and TO-BE process flows), Business Rules and Validation Logic, User Roles and Authorization Requirements, Reporting and Output Requirements, Integration Points with other SAP modules or external systems, Data Requirements, and Open Items and Assumptions.

IBM Bob can generate all of these sections when provided with appropriate input context.

Approach 1

Generate FS from Business Requirements

The most common approach is to feed Bob the business requirements gathered during workshops and ask it to produce the FS. The consultant attaches the requirements document (Excel, Word, or plain text) using a context mention, then prompts Bob to generate the full FS.

Example Prompt:

```
Using @/requirements/MM-Procurement-Workshop-Notes.docx as input, generate a complete SAP Functional Speci
```

Bob reads the workshop notes, extracts the relevant requirements, applies the FS template structure, and writes the complete Word document to the specified path. The consultant then reviews, refines, and approves.

Approach 2

Generate FS from a WRICEF Log

When a WRICEF (Workbooks, Reports, Interfaces, Conversions, Enhancements, Forms) list exists, Bob can iterate over each item and produce a draft FS for each one. By referencing the WRICEF log as a context file and instructing Bob to process each row, teams can produce a full set of draft specifications in a single session.

Example Prompt:

```
Read @/project/WRICEF-Log.xlsx. For each Enhancement row in the sheet, generate a separate Functional Spec
```

Approach 3

Generate FS from Existing ABAP Program (Reverse FS)

Legacy SAP systems frequently have custom ABAP objects whose original Functional Specification was never written or has been lost. IBM Bob can read the ABAP source code and produce a reverse-engineered Functional Specification that documents what the program does from a business perspective — an invaluable artifact for system upgrades and S/4HANA migration projects.

Example Prompt:

```
Read @/abap/ZPURCHASE_ORDER_EXIT.abap. This is a legacy ABAP user exit. Generate a Functional Specificatio
```

Generating technical specifications with IBM Bob

A Technical Specification (TS) in SAP describes how a development object will be implemented — covering data structures, program logic, database access, interfaces, and performance considerations. IBM Bob is particularly effective at Technical Specification generation because it can read and reason about actual ABAP source code.

Standard sections of an SAP technical specification

A complete SAP Technical Specification contains: Document Header, Reference to Functional Specification, Technical Overview, Program Type and Object Name (Report, Function Module, BAPI, Enhancement, BAdI, etc.), Data Dictionary Objects (Tables, Structures, Data Elements), Program Logic Description (step-by-step algorithmic walkthrough), Selection Screen Design, ALV/Output Layout (for reports), Error Handling and Exception Management, Performance Considerations, Authorization Objects, Transport Strategy, and Unit Test Approach.

Approach 1

Generate TS from ABAP Source Code

IBM Bob's most powerful TS generation capability is reading an ABAP program and producing a full Technical Specification. The developer provides the ABAP source using a context mention, and Bob systematically documents every aspect of the implementation.

Example Prompt:

```
Read @/src/abap/ZFI_AR_AGING_REPORT.abap. Generate a complete SAP Technical Specification document for thi
```

Approach 2

Generate TS from Functional Specification

When an ABAP object is yet to be developed, Bob generates a Technical Specification by reading the Functional Specification and expanding it into the technical domain. The resulting TS serves as the development blueprint, which the ABAP developer then implements.

Example Prompt:

```
Read @/docs/FS-SD-CUSTOMER-CREDIT-CHECK-v1.0.docx. Based on this Functional Specification, design and writ
```

Approach 3

Generate TS for SAP Interface (IDoc, RFC, API)

Interface design documents are among the most complex to produce. Bob can analyze interface source code or specification inputs and generate a complete Interface Technical Design Document covering: source and target system, message format (IDoc, RFC, REST/SOAP), field mapping tables, error handling and retry logic, and monitoring approach.

Example Prompt:

```
Read @/interfaces/SAP_to_Salesforce_CustomerMaster_Mapping.xlsx. Generate a full Interface Technical Desig
```

Additional SAP document types IBM Bob can generate

IBM Bob's document generation capability extends well beyond Functional and Technical Specifications. The following table summarizes the full range of SAP project artifacts that Bob can produce.

Table: SAP document types IBM Bob can generate, by project phase

Document Type	SAP Phase	Bob's Input Source
Functional Specification (FS)	Explore / Realize	Workshop notes, requirements workbook, WRICEF log, ABAP code
Technical Specification (TS)	Realize	ABAP source, FS document, interface mappings
Business Process Procedure (BPP)	Explore	Process flow diagram, SAP config, workshop notes
Configuration Workbook	Explore / Realize	Business requirements, SAP module scope
Data Migration Strategy	Realize	Legacy data extracts, field mapping sheets, transformation rules
Interface Design Document (IDD)	Realize	Interface inventory, field mapping tables, API specs
Unit Test Script	Realize	Functional Specification, configuration workbook
Integration Test Plan	Realize / Deploy	Process flows, interface dependencies, test scenarios
User Acceptance Test (UAT) Script	Deploy	Business process descriptions, BPPs, user stories
Cutover Plan	Deploy	Sequence of cutover activities, system dependencies, go-live criteria
End-User Training Material	Deploy	BPPs, system screenshots, user guides
Knowledge Transfer Document	Run	Technical Specifications, configuration workbooks, runbooks
ABAP Code Review Report	Realize	ABAP source code, transport log

Business process procedures

A business process procedure (BPP) is a step-by-step guide describing how a user performs a specific business task in SAP. Bob generates BPPs by combining knowledge of the SAP standard process with project-specific configuration details. When referenced with workshop notes and config documents, Bob produces accurate, numbered step-by-step procedures including transaction codes, screen navigation, field values, and decision points.

Data migration documents

Data migration is one of the highest-risk areas of any SAP implementation. IBM Bob can generate: Data Migration Strategy documents, Legacy-to-SAP field mapping specifications, Data Quality Assessment reports (by analyzing extract samples), LSMW/BAPI/LTMC migration object specifications, and Migration Reconciliation and sign-off templates.

ABAP code review and quality reports

Bob can read an ABAP source file and produce a structured Code Review Report identifying: performance anti-patterns (for example, SELECT * inside loops, missing WHERE clauses), missing authorization checks, hard-coded values that should be customizing entries, non-compliance with SAP coding standards, and suggested refactoring actions with rationale.

Setting up IBM Bob for SAP projects

A one-time setup is required to configure IBM Bob optimally for SAP documentation generation. This section describes the recommended configuration.

Workspace structure

Organize the project workspace so Bob can navigate it efficiently. The recommended folder structure is: /abap (ABAP source files exported from SAP), /docs (generated and approved documentation), /templates (FS, TS, BPP, and other document templates), /requirements (workshop notes, requirements workbooks, WRICEF log), /interfaces (field mapping sheets, interface inventories), and /.bob (Bob configuration — rules and custom modes).

Configuring SAP-specific rules

Create a rules file at .bob/rules/sap-standards.md with project-specific instructions. Example content: document naming convention (FS-<Module>-<Object>-v<Version>.docx), required sections for each document type, SAP module abbreviations (MM, SD, FI, CO, PP, WM, EWM, PS, and other module abbreviations), language standard (formal British English for client-facing documents), and reference to the SAP Activate methodology for phase context.

Creating a custom SAP consultant mode

Define a custom mode in .bob/custom_modes.yaml to give Bob the persona of an SAP functional or technical consultant. The mode should define the role (e.g., 'Senior SAP Functional Consultant with 15 years of experience across MM, SD, and FI modules'), behavioral instructions (always reference SAP Activate, always produce complete documents rather than outlines, always include an Open Items section), and tool permissions (file read/write, terminal for transport log retrieval).

Prompt engineering guide for SAP documentation

The quality of Bob's output is directly proportional to the quality of the prompt. This section defines recommended practices for prompting Bob in SAP documentation scenarios.

The CARE prompting framework

For SAP documentation prompts, apply the CARE framework:

- **Context (C):** Provide the SAP module, project phase, and nature of the object (e.g., 'This is an SAP MM Procurement enhancement for a Realize-phase WRICEF item').
- **Action (A):** Specify the exact document to be produced and the target file path (e.g., 'Generate a complete Technical Specification and save it as docs/TS-MM-ZENH001.docx').
- **References (R):** Attach all relevant source files using @ mentions (e.g., '@/abap/ZENH001.abap, @/docs/FS-MM-ZENH001.docx, @/templates/TS-Template.docx').
- **Expectations (E):** Define the required sections, depth of coverage, and any special instructions (e.g., 'Include all SELECT statements, document every BAdI method, list all SAP notes referenced in the code').

Iterative refinement

IBM Bob supports iterative document refinement within the same conversation. After the initial document is generated, the consultant can follow up with targeted refinements such as: 'Add a process flow diagram description to the third section', 'Expand the error handling section to include SAP message class details', or 'Update the document header with Author: John Smith, Version: 1.1, Date: 2025-01-15'. Bob reads the previously generated file and applies only the requested changes.

Batch document generation

For large SAP projects with dozens of custom objects, Bob can be prompted to iterate over a WRICEF list and generate all documents in a single session. Using Plan mode to design the generation strategy and Agent mode to execute it, a project team can produce a complete documentation pack in hours rather than weeks.

Quality assurance and review workflow

IBM Bob produces draft documents. A human review and approval workflow must be in place before any document is baselined or shared with the client.

Recommended review process

The recommended four-step review process is:

1. Bob generates the draft document and saves it to the /docs/draft/ folder.
2. The assigned consultant opens the draft, reviews it against the source inputs, and marks corrections using track changes.
3. Bob incorporates the tracked corrections in a follow-up prompt.
4. The Lead Consultant or Solution Architect approves the document and moves it to /docs/approved/ with the version number incremented.

What Bob gets right and what to verify

Bob is highly reliable for: document structure and completeness, extraction of logic from ABAP code, field-level mapping documentation, and standard SAP terminology. Human reviewers should pay particular attention to: business context and intent (Bob interprets code, not business intent), project-specific naming conventions, client-specific business rules not present in the source files, and authorization and security requirements that are organizationally governed.

Case studies and sample outputs

Case study: S/4HANA migration — ABAP custom code documentation

Scenario: A large manufacturing client was migrating from SAP ECC 6.0 to SAP S/4HANA 2023. The legacy system had 847 custom ABAP objects, of which only 312 had existing documentation. The project team used IBM Bob to generate reverse-engineered Technical and Functional Specifications for all 535 undocumented objects over a two-week period.

Approach: ABAP source files were exported from the ECC system and placed in the /abap workspace folder. A custom SAP Consultant mode was configured with ECC-to-S/4HANA migration context. Bob iterated through the /abap folder, generating a TS and FS pair for each object.

Outcome: 535 complete documentation pairs were generated as draft documents. After human review and approval, 489 were accepted with minor edits, 38 required moderate revision, and 8 required significant rework due to missing business context. The team estimated a 70% reduction in documentation effort compared to manual authoring.

Case study: SAP Activate Explore phase — Functional specification pack

Scenario: A retail client's SAP SD and MM implementation required 64 Functional Specifications to be produced in a six-week Explore phase. The team of four consultants used IBM Bob to draft all 64 documents from workshop notes and the WRICEF log.

Approach: Workshop notes were organized in /requirements by module and date. The WRICEF log was stored at /project/WRICEF-v3.xlsx. Consultants ran batched prompts by module (SD enhancements, MM reports, FI interfaces) and refined the outputs in follow-up conversations.

Outcome: All 64 Functional Specifications were drafted within three weeks, giving consultants three weeks for review, client sign-off, and revision — a workflow that would not have been possible with manual authoring alone.

Summary and next steps

IBM Bob represents a significant productivity multiplier for SAP project teams. Its ability to read ABAP source code, process structured input files, follow project-specific rules and templates, and autonomously generate complete Word documents in a single agentic session fundamentally changes the economics of SAP documentation production.

The key capabilities covered in this publication are:

- Functional Specification generation from requirements, WRICEF logs, and reverse-engineering of ABAP code
- Technical Specification generation from ABAP source, FS documents, and interface mappings
- Full SAP project artifact coverage: BPPs, configuration workbooks, data migration documents, test scripts, cutover plans, and training materials
- Custom mode and rules configuration for consistent, team-wide documentation standards
- Prompt engineering patterns that maximize the quality and completeness of generated documents

SAP teams adopting IBM Bob should take the following recommended steps:

- Install IBM Bob in VS Code and run the Getting Started tutorial to familiarize the team with the interface.
- Configure the .bob/rules/sap-standards.md file with project conventions and document templates.
- Create a custom SAP Consultant mode for the project team.
- Run a pilot with 5–10 ABAP objects or requirements items to calibrate the prompts and templates for the specific engagement.
- Scale to the full WRICEF list once the pilot quality threshold is met.

Sample prompt library

The following curated prompts can be used as-is or adapted for your SAP project.

Generate functional specification from workshop notes

```
"Read @/requirements/<workshop-file>. Generate a complete SAP Functional Specification for <Object Name> i
```

Generate technical specification from ABAP code

```
"Read @/abap/<ProgramName>.abap. Generate a full SAP Technical Specification. Include: Program overview, a
```

Generate business process procedure

```
"Using @/requirements/<process-notes> and @/config/SAP-<Module>-Config-Workbook.xlsx, generate a Business
```

Generate data migration field mapping

```
"Read @/migration/<LegacyExtract-FieldList.xlsx>. Map each legacy field to the corresponding SAP S/4HANA f
```

Generate ABAP code review report

```
"Read @/abap/<ProgramName>.abap. Perform a technical code review. Identify and document: performance issue
```

Appendix B: Recommended document templates

IBM Bob uses templates stored in the /templates folder as a reference when generating documents. The following templates are recommended as a minimum set for SAP project deployment:

- FS-Template.docx: Standard SAP Functional Specification template with all required sections, corporate branding, and version control header.
- TS-Template.docx: Standard SAP Technical Specification template with ABAP-specific sections, table documentation blocks, and transport details.
- BPP-Template.docx: Business Process Procedure template with step numbering, transaction code column, screenshot placeholder, and expected results section.
- ITDD-Template.docx: Interface Technical Design Document template covering source system, target system, message format, field mapping table, and error handling.
- FieldMapping-Template.xlsx: Excel template for data migration field mappings with pre-formatted headers and validation drop-downs for transformation rules.
- TestScript-Template.xlsx: UAT test script template with test case ID, description, prerequisite, steps, expected result, actual result, and pass/fail columns

Integrations and Operation

IBM Bob is a capable code-generation and review tool when working from the knowledge already in its training data. It becomes a different class of tool entirely when it can read your actual project context — the ServiceNow change request that describes what must be built, the Signavio process model that governs how it should behave, the GitHub repository that holds what was built before, the LeanIX architecture record that defines what the system is part of. That gap between a well-informed response and a well-reasoned one is filled by integration.

Enterprise SAP delivery is a multi-system workflow. A task begins in a project management platform. It references design documents held in a knowledge repository. It produces code that travels through version control and lands in a transport. It closes back in a change management platform with a completion record. At each handoff, context is lost — because the developer must mentally translate across system boundaries, holding in their head what each system knows and reconciling the differences. IBM Bob, connected to all of those systems through a common integration framework, can hold that context automatically, carry it through the workflow without loss, and surface exactly the right information at each step.

This section describes how that integration framework is built. It covers the Model Context Protocol as the architectural standard that makes tool connections uniform, the authentication patterns that keep enterprise data secure, the human-approval governance model that ensures AI assistance remains accountable, and the detailed implementation of each major integration — ServiceNow, GitHub, Tricentis, Signavio, LeanIX, and SAP BTP. The principles are consistent. The implementations are specific. Together they give a delivery team the reference architecture needed to deploy IBM Bob not as a stand-alone AI assistant, but as a connected participant in the full enterprise delivery lifecycle.

What this section covers

This section follows the integration stack from the architectural foundation to each individual tool connection. The topics build on each other, but each can be read independently by teams who have already made progress on specific integrations.

The integration imperative and MCP architecture opens the technical content. It establishes why Bob cannot produce its best output without external context, then introduces the Model Context Protocol — the USB-standard

interface that makes tool connections composable, replaceable, and governable. The four-layer enterprise integration landscape (knowledge repositories, operational systems, development tools, and data layer) provides the structural framework for everything that follows.

ServiceNow integration covers the most common change management connection in enterprise SAP programs. This section documents JWT and OAuth 2.0 authentication patterns, the MCP server implementation that exposes ServiceNow tickets, knowledge articles, and CMDB records to Bob, and the write-back pattern that closes the audit loop after Bob completes a review or generates a document.

GitHub and CI/CD integration addresses both the version control connection — reading repository history, patterns, and existing implementations — and the automated pipeline integration that allows Bob to participate in pull request review, code quality gates, and deployment workflows for both ABAP transports and UI5 applications. The section covers connecting Bob to GitHub with Personal Access Tokens and the GitHub MCP Server, step-by-step CI/CD examples for SAP UI5 (OpenUI5) projects and for ABAP projects managed with abapGit, GitHub Actions pipeline templates for both, and deployment of SAP CAP and ABAP applications to SAP BTP Cloud Foundry through SAP Continuous Integration and Delivery. A quick-reference card lists every Bob Git command, slash command, and natural-language trigger.

Tricentis Tosca integration documents the connection between Bob and the primary test management platform used on SAP programs. This section covers the ToscaCloud MCP server, Bob-assisted test case generation from functional specifications, and the feedback loop that closes defect analysis back into the development workflow. It also describes the three-layer issue resolution architecture (Issue Logging, Test Automation, and AI Resolution), in which Bob reads issues from Jira and ServiceNow and corrects them directly in Tricentis test scripts. Further topics are descriptive programming and how Bob accelerates function module identification, CI/CD integration with Jenkins for automated test delivery, and a before-and-after comparison of the productivity impact.

Signavio and LeanIX integration pairs the two tools that provide process and architecture context. Signavio supplies the business process models that govern what code must do. LeanIX supplies the application portfolio data that defines what the system is part of. Both integrations are documented with reference implementations and the specific prompt patterns that make them productive. The section covers two integration patterns, browser automation and a custom MCP server, and helps you choose between them. It shows how to build and register the signavio-mcp server, including authentication, region-specific endpoints, and secure credential handling, with a worked example in which Bob creates a Signavio journey model through conversation. It then shows how to build a LeanIX MCP server using OAuth 2.0 and GraphQL, and closes with an example of how the combined integration changes the developer and architect experience.

SAP BTP integration closes the tool-specific coverage, documenting the SAP-released MCP server for BTP services and the Bob workflow patterns for Integration Suite, BAS, and Build that depend on it.

Playwright integration brings browser automation into Bob's working context. This section documents the Playwright MCP server, which lets Bob drive a real browser through structured accessibility snapshots instead of screenshots, so it can navigate, inspect, and interact with SAP Fiori and UI5 applications predictably. It covers using Bob to write end-to-end test scripts that rely on stable UI5 control identifiers, to investigate defects by reproducing them in a live session, and to run smoke tests after a deployment. Playwright is positioned as the developer-facing complement to Tosca's enterprise test management, not a replacement for it.

Jira integration covers the agile delivery connection for programs that track work in Atlassian rather than ServiceNow. This section documents OAuth 2.0 and API token authentication, the Atlassian MCP server that exposes epics, user stories, acceptance criteria, sprint boards, and Confluence pages to Bob, and the traceability pattern that links generated code, test cases, and review findings back to the originating story. Prompt patterns show how Bob turns acceptance criteria into technical designs and posts structured status updates back to the ticket, so the delivery record stays current without manual effort.

SharePoint integration addresses the knowledge repository layer where most SAP programs keep their unstructured documentation: functional specifications, solution design documents, blueprints, and sign-off records. This section documents the Microsoft Graph-based MCP server, Microsoft Entra ID authentication with least-privilege site-scoped permissions, and the retrieval patterns that let Bob find and reference the authoritative version of a

document instead of a stale copy. It also covers the governance controls that keep document access consistent with existing SharePoint permissions, which matters in regulated environments.

SAP Cloud ALM integration connects Bob to the application lifecycle management platform that coordinates SAP implementation and operations. This section documents an MCP server built on the SAP Cloud ALM public APIs, authenticated through a BTP service key with OAuth 2.0 client credentials. Through it, Bob can read projects, requirements, user stories, features, process hierarchies, and test plans. Reference workflows show how Bob checks development artifacts against Cloud ALM requirements, generates test cases aligned to the process hierarchy, and writes results back, so the lifecycle record stays the single source of truth across build, test, and deploy.

Building a custom MCP server with IBM Bob lets you connect Bob to systems that no ready-made MCP server supports. Bob helps design, code, and test the server itself, so the connection can be built quickly and in a controlled way. Once the server is in place, Bob can use those systems in its everyday work.

This section also covers how to troubleshoot issues that arise while you configure an MCP server. It explains how to check the configuration file, credentials, connectivity, and server logs, so you can fix the setup quickly when Bob can't connect or doesn't see the server's tools.

The code examples and reference implementations throughout this section are provided as working foundations. They should be reviewed and adapted to meet your organization's security, governance, and operational standards before moving into production.

Value across enterprise roles

Enterprise integrations are not a developer concern alone. The decision to connect IBM Bob to ServiceNow, GitHub, or LeanIX is an architectural and governance decision — one that affects what every role on the program can do and how accountable the AI-assisted workflow is.

Table: IBM Bob integration value by enterprise role.

Role	What integration changes
Senior Leadership & PMO	Integration closes the audit trail that isolated AI assistance leaves open. When Bob reads a ServiceNow change request before generating code, the CR number travels into the ABAP header, the transport, and the review record automatically. Leadership gets a verifiable chain from requirement to delivered code without relying on manual cross-referencing by developers under delivery pressure. Connected integrations also compress the context-gathering step that currently adds unbillable hours to every development task.
Enterprise & Solution Architects	LeanIX and Signavio integrations give Bob architecture-aware context at the point of code generation. An architect's decision — that a particular service boundary should not be crossed, that a CDS view is shared across domains — is encoded once in the architecture repository and surfaced to Bob automatically when that scope is relevant. Impact analysis improves because Bob can cross-reference a proposed change against the live architecture model before the change is implemented, not after it has already been transported to quality.
Project & Delivery Managers	GitHub integration means Bob can participate in code review automatically — surfacing comments aligned to project standards, checking for missing documentation headers, flagging deviations from the agreed Clean Core boundary — without consuming a senior developer's review bandwidth for every change. CI/CD pipeline integration means the same quality gates that apply to human-authored code apply to Bob-generated code, with no special treatment. The consistency that results is directly measurable in defect rates and rework cycles.
ABAP & Full-Stack Developers	Integration means the context a developer would previously gather manually — reading the CR, checking the design doc, reviewing the existing pattern — arrives in the conversation before the first prompt is written. Bob reads the live ServiceNow ticket, fetches the Signavio process model, checks the GitHub repository for prior implementations, and arrives at the code generation step already informed. The developer's job is to verify the context is correct and approve the output, not to translate across five systems before writing a line of code.

The compounding effect here is stronger than in any single-tool scenario. A developer who does not have to gather context manually writes faster. Code that is informed by the actual architecture model has fewer defects. Defects that are caught by automated CI/CD gates before transport never consume QA time. Audit records that are written back to ServiceNow automatically never require a project manager to chase completion evidence. Each integration removes one class of friction. Together they remove the coordination overhead that currently scales linearly with project complexity — and that is the outcome that justifies the integration investment at program level, not just at developer level.

External tool integration overview

One standard interface: Model Context Protocol

Until now, we have seen how IBM Bob connects to SAP — each path offering a different trade-off between speed, governance, and capability. But SAP is rarely an island. In the real world, a developer's day spans many systems: raising a change in ServiceNow, linking a defect in Jira, pulling an order status from an SAP OData service, logging the resolution in Cloud ALM. Each of these systems speaks its own language, and without a common interface, every integration is a one-off effort. This is the problem that the Model Context Protocol (MCP) solves. It provides a single, standardized interface through which IBM Bob can speak to any external system. Instead of building a bespoke connector for each tool, each service is exposed as an MCP server. The agent speaks one protocol; every system plugs in the same way — governed, reusable, and AI-native.

The story: From a single ask to many systems

Imagine a developer types a single natural-language request into IBM Bob:



Figure: The single natural-language request given to IBM Bob

In a pre-MCP world, that one sentence would force the developer to navigate four separate tools, log in to four separate systems, and manually copy data between them. With MCP, IBM Bob takes the single instruction, plans a sequence of actions, and calls each connected MCP tool in turn — all without the developer leaving the conversation. Here is the architecture that makes that possible:

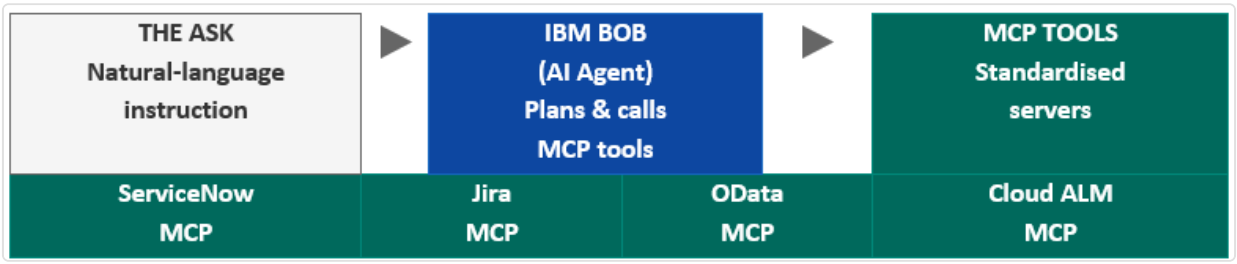


Figure: MCP architecture, one ask dispatched to many systems

How MCP works in IBM Bob

MCP is not a product — it is a protocol. When IBM Bob is configured with MCP servers, each server registers a set of self-describing tools that Bob can discover and call. Bob can read what each tool does, what parameters it needs, and what it returns. It does not need to be pre-programmed with knowledge of each system: it reasons about which tool to call based on the developer's intent, and chains multiple calls together to fulfill a complex request.

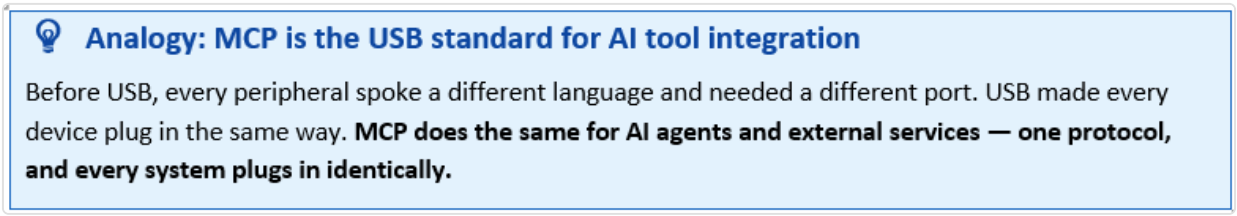


Figure: MCP as the USB standard for AI tool integration

The flow from intent to result is straightforward:

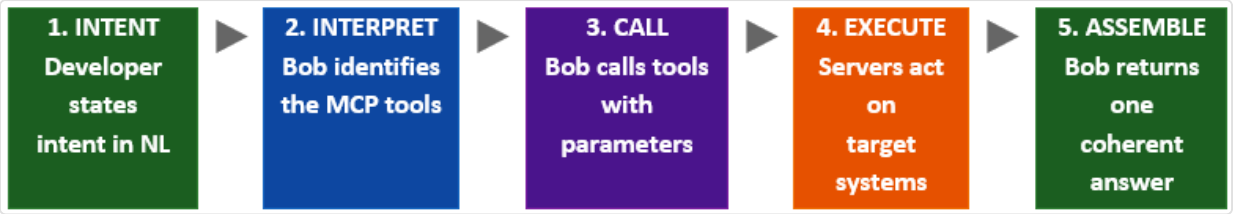


Figure: The five-step MCP flow from intent to result

> **Note:** Steps 3 and 4 can run in sequence or in parallel where the actions are independent.

MCP core concepts

Concept	Description
MCP Host	The application running the AI model — IBM Bob
MCP Client	Built into Bob — manages all server connections
MCP Server	A program exposing tools, resources, or prompts over MCP
Tools	Callable functions Bob can invoke (for example, create_issue)
Resources	Data sources Bob can read by URI
Prompts	Pre-defined prompt templates registered by servers

How MCP works — Step by step

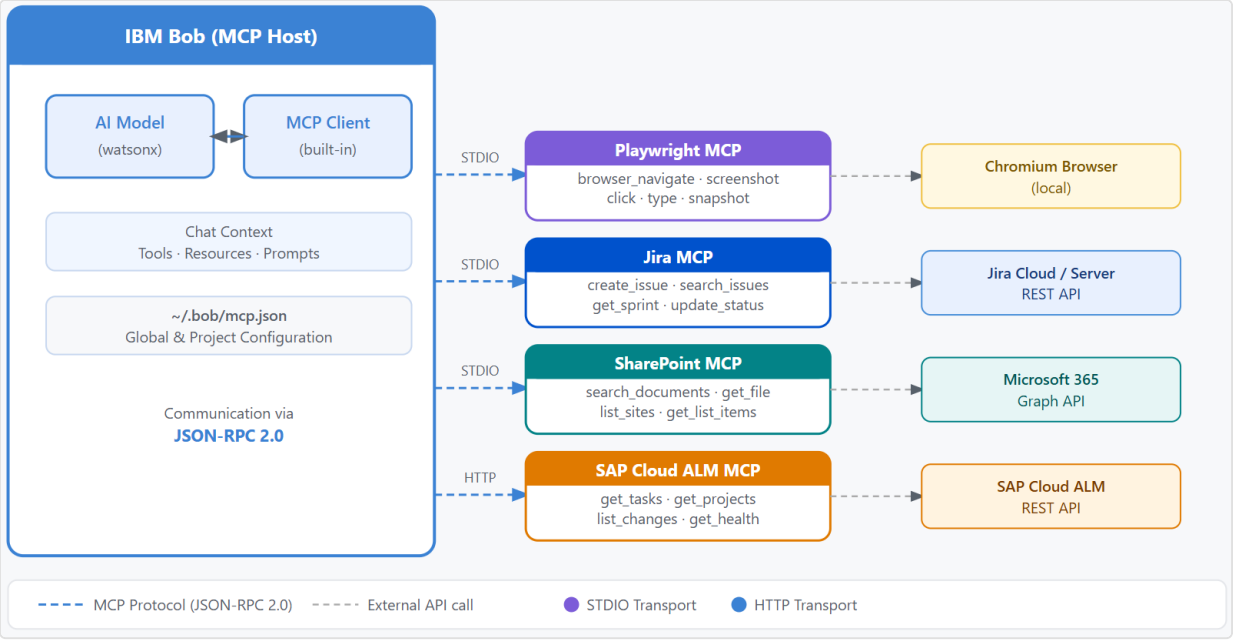


Figure: MCP Architecture — IBM Bob & MCP Servers

1. You send a request to Bob — e.g. "Find all open Jira tickets for Sprint 42"
2. Bob consults available tools — knows the Jira MCP server has `search_issues`
3. Bob calls the tool — sends a JSON-RPC 2.0 message to the server
4. The MCP server executes — calls the Jira REST API with your credentials
5. Results returned to Bob — structured data flows back over the protocol
6. Bob uses the data — incorporates results into its response
7. Transport Types MCP supports different transport mechanisms for communication between the MCP client and MCP server. The choice of transport depends on the deployment model, communication requirements, and environment.

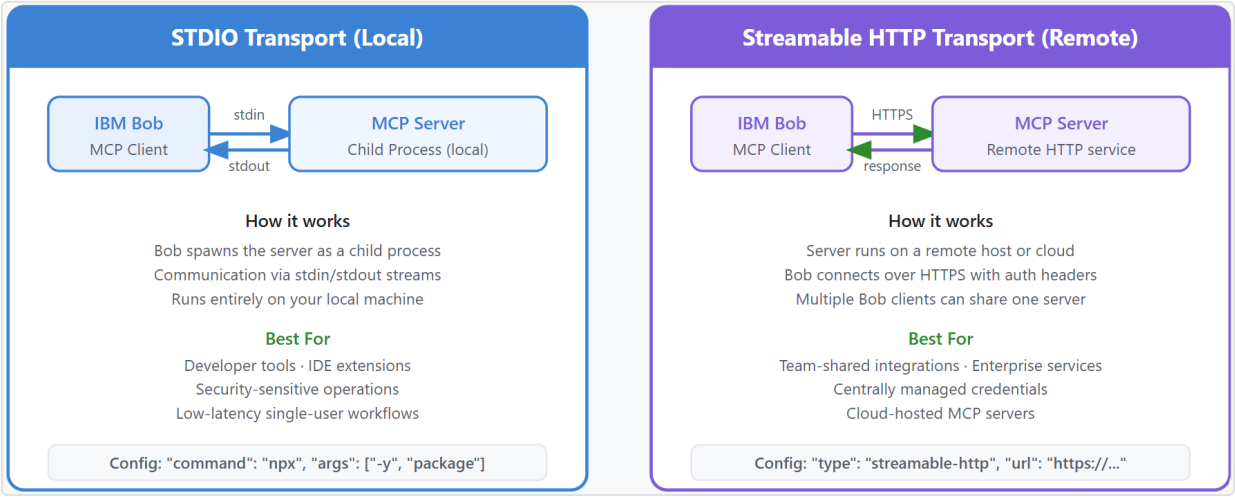


Figure: MCP Transport Types — STDIO vs HTTP

STDIO Transport (Local) The MCP server runs as a child process on your local machine, communicating via stdin/stdout. Best for local tools, IDE integrations, developer utilities.

```
{
  "mcpServers": {
    "my-local-server": {
      "command": "node",
      "args": ["./build/index.js"]
    }
  }
}
```

Figure: Example configuration for a local STDIO MCP server

Streamable HTTP Transport (Remote) The server runs as a remote HTTP service. Best for shared team servers and cloud-hosted enterprise integrations.

```
{
  "mcpServers": {
    "my-remote-server": {
      "type": "streamable-http",
      "url": "https://mcp.yourcompany.com/endpoint",
      "headers": { "Authorization": "Bearer ${env:MCP_TOKEN}" }
    }
  }
}
```

Figure: Example configuration for a remote Streamable HTTP MCP server

Configuration in IBM Bob

Level	File Path	Scope
Global	~/.bob/mcp.json	All workspaces
Project	.bob/mcp.json in project root	Current project only; committable to version control

> **Note:** Project-level configuration takes precedence over global when server names conflict.

MCP in SAP delivery context

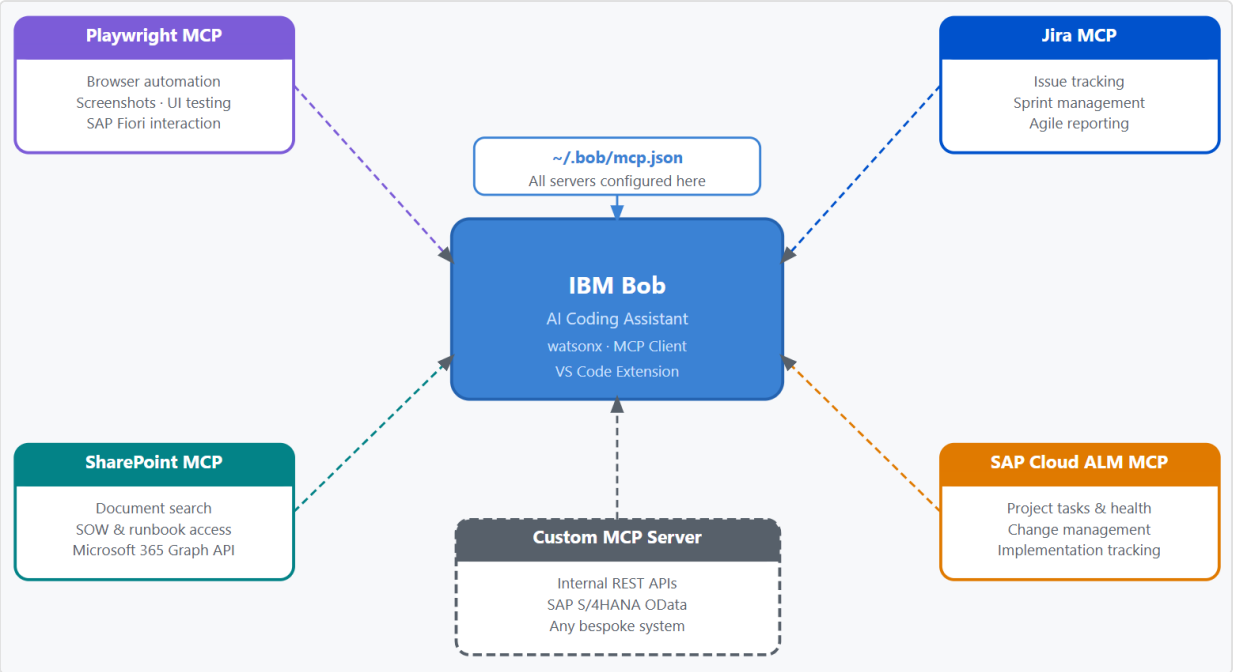


Figure: IBM Bob connected to MCP servers for SAP delivery

MCP Integration	What IBM Bob Can Do
Playwright	Automate SAP Fiori UI testing, screenshots, form interaction
Jira	Create issues, query sprints, update story status
ServiceNow	Create and update incidents, query requests, retrieve CMDB and ticket information
SharePoint	Search project documents, retrieve SOWs and runbooks
SAP Cloud ALM	Query tasks, check health, retrieve change requests
SAP BTP	Interact with BTP services, query application and environment information, and automate platform operations
Custom MCP Server	Connect to any internal REST API or SAP OData service

> **Note:** These are just a few examples of MCP integrations. Many more integrations can be added based on the project's specific needs, requirements, and the tools or systems used in the environment.

Finding and installing MCP servers

IBM Bob does not ship with pre-installed MCP servers. There are three pathways to add MCP capabilities:

1. **Bob Marketplace**: search and install servers directly from inside Bob's settings panel
2. **External MCP Registries**: browse the broader MCP ecosystem for community and third-party servers
3. **Build Your Own**: create a custom MCP server when no existing server meets your needs

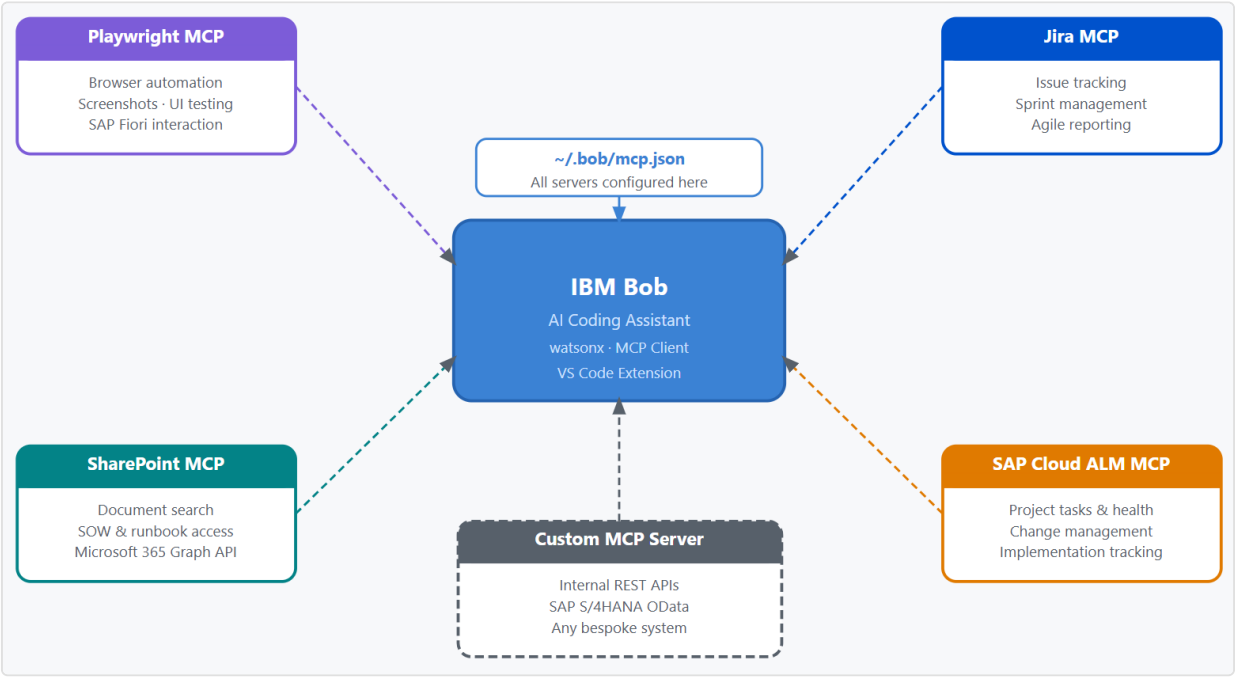


Figure: IBM Bob connected to example MCP servers

Pathway 1 — Bob Marketplace

Accessing the Marketplace

1. Open the **IBM Bob** panel in VS Code
2. Click the **Settings** icon (⚙️)
3. Select the **MCP** tab
4. The marketplace listing appears below your currently configured servers

Searching for an MCP Server

- Use the **search box** in the MCP tab to filter by name or keyword
- Browse by category — productivity, databases, cloud platforms, testing tools
- Each listing shows the server name, description, publisher, and an **Install** button

Installing from the Marketplace

1. Click **Install** on the desired server
2. Bob prompts for required settings (API keys, URLs, etc.)
3. Once configured, the server appears in your active MCP list and connects automatically

Uninstalling a Marketplace Server

1. Locate the installed server in the marketplace view
2. Click **Uninstall**
3. Bob removes it from your configuration file immediately

Pathway 2 — External MCP registries

When the Bob Marketplace does not have what you need, check these external sources:

Source	URL	What It Offers
MCP Market	https://mcpmarket.com	Largest registry — 48,000+ servers, categorised by type (Collaboration, Cloud, DevOps, Browser Automation, etc.), Official/Featured rankings, leaderboard
MCP.so	https://mcp.so	Community registry with broad coverage
Smithery	https://smithery.ai	Curated marketplace with quality ratings
GitHub — Official Servers	https://github.com/modelcontextprotocol/servers	Official reference implementations from the MCP team
npm	https://npmjs.com	Search @modelcontextprotocol or mcp-server-* for packages

Recommended Search Strategy

When looking for an MCP server for a specific tool or service, follow this order:

1. **Bob Marketplace first**: Settings → MCP tab → search by name

2. **MCP Market** (https://mcpmarket.com): largest catalog, filter by category (e.g. "Collaboration Tools" for SharePoint/Teams, "Cloud Infrastructure" for SAP BTP)
3. **Smithery** (https://smithery.ai): for quality-rated, vetted servers
4. **MCP.so**: broader community coverage
5. **GitHub** (https://github.com/modelcontextprotocol/servers): official reference implementations
6. **npm**: search for packages if you have a specific package name in mind
7. **Build custom**: when nothing suitable exists (see Building a Custom MCP Server)

> **Security note:** Always review the source code and publisher reputation before installing community MCP servers. Run them with minimum required permissions.

Manual Configuration — Installing Any MCP Server

When you find a server outside the marketplace, register it manually in your `mcp.json`.

Step 1 — Open the Config File

Option A — via Bob UI

1. **Settings** (⚙️) → **MCP** tab
2. Click **Edit Global MCP** (all workspaces) or **Edit Project MCP** (this project)

Option B — direct update

- Global: `~/.bob/mcp.json`
- Project: `<project-root>/.bob/mcp.json`

Step 2 — Add the Server Entry

npm / npx-based server:

```
{
  "mcpServers": {
    "my-server": {
      "command": "npx",
      "args": ["-y", "mcp-server-package-name"]
    }
  }
}
```

Locally built Node.js server:

```
{
  "mcpServers": {
    "my-server": {
      "command": "node",
      "args": ["/absolute/path/to/build/index.js"],
      "env": {
        "API_KEY": "${env:MY_API_KEY}"
      }
    }
  }
}
```

Remote HTTP server:

```
{
  "mcpServers": {
    "my-remote-server": {
      "type": "streamable-http",
      "url": "https://mcp.yourcompany.com/endpoint",
      "headers": {
        "Authorization": "Bearer ${env:MCP_TOKEN}"
      }
    }
  }
}
```

Step 3 — Save and Verify

- Save the file — Bob hot-reloads automatically
- Check the **MCP** tab to confirm the server shows as **connected**

Managing installed servers

Enable / Disable a Server

Settings → **MCP** tab → toggle the power switch next to the server.

Useful for temporarily disabling a server without removing its configuration.

Restart a Server

Settings → **MCP** tab → click the **Restart** icon (🔄) next to the server.

Use this after changing environment variables or updating the server binary.

Manage Individual Tools

Each MCP server can expose many tools. You can selectively enable or disable them:

1. **Settings** → **MCP** tab
2. Expand the server entry
3. Toggle individual tools on or off

> **Why disable tools?** Each enabled tool definition consumes context window space. Disabling unused tools leaves more room for code and conversation history.

Set Network Timeout

Use the following settings for slow remote servers:

1. **Settings** → **MCP** tab → find the server
2. Set **Network timeout** (30 seconds to 5 minutes; default 1 minute)

Auto-Approving MCP Tools

By default, Bob asks for your confirmation before calling any MCP tool.

Global Auto-Approve

1. **Settings** → **Auto-approve** section
2. Enable **Use MCP servers**

> This is a master switch. If disabled, no MCP tools auto-approve regardless of per-tool settings.

Per-Tool Auto-Approve via Config

```
{
  "mcpServers": {
    "my-server": {
      "command": "npx",
      "args": ["-y", "my-mcp-package"],
      "alwaysAllow": ["tool_name_1", "tool_name_2"]
    }
  }
}
```

Troubleshooting MCP servers quick reference

Problem	Likely Cause	Fix
Server not appearing	JSON syntax error in mcp.json	Validate JSON — check commas, brackets, quotes
"Command not found"	Runtime not on PATH	Verify node/npx is installed; check PATH
Stuck on "Connecting…"	Package download issue	Run npx -y <package> in a terminal to test
401 / 403 errors	Wrong or missing credentials	Check env values; verify API key is valid
Tools not visible	Server connected but tools not loaded	Click Restart (🔄) in the MCP tab
Frequent disconnects	Network timeout too short	Increase timeout in MCP settings

Summary

Task	How
Search for MCP servers	Settings → MCP tab → Marketplace section
Install from marketplace	Click Install, follow prompts
Add a server manually	Edit mcp.json, add entry under mcpServers
Disable without deleting	Toggle power switch in MCP tab
Auto-approve specific tools	Set alwaysAllow in config or toggle in MCP tab

Playwright MCP

Why would an AI assistant need access to Playwright?

Imagine you are testing an SAP Fiori application. Instead of manually opening the browser, navigating through screens, entering data, clicking buttons, and verifying the results, you can ask IBM Bob:

"Open the SAP Fiori application, create a sales order, and verify that it was created successfully."

For an AI assistant such as IBM Bob to perform these UI interactions, it needs a way to control and interact with the browser. This is where **Playwright MCP** becomes valuable.

Playwright MCP exposes browser capabilities as MCP tools. Bob can discover these tools and use them to navigate web pages, inspect page content, click elements, enter information, take screenshots, and validate results.

The interaction can be summarized with the following interaction chain:

```
User Request
     ↓
   IBM Bob
     ↓
Playwright MCP
     ↓
Web Browser
     ↓
UI Interaction
     ↓
Result / Screenshot
     ↓
   IBM Bob
     ↓
User Response
```

Key takeaways

Playwright MCP enables an AI assistant to interact with web applications through the browser, turning natural-language instructions into automated UI actions and validation.

Overview

The **Playwright MCP server** gives IBM Bob the ability to control a real web browser. Bob can navigate pages, interact with UI elements, take screenshots, fill forms, and extract data from live web applications.

The following list describes possible SAP delivery use cases:

- Automated browser-based UI testing of SAP Fiori applications
- Taking screenshots of SAP systems for documentation and sign-off
- Navigating SAP web portals to retrieve data
- Regression testing UI behavior after transport deployments

Prerequisites

Requirement	Details
IBM Bob IDE	Installed and running
Node.js	v18 or later — node --version
npx	Bundled with Node.js — npx --version

> No manual package installation required. `npx -y` downloads and runs automatically on first use.

3 key tools exposed

Tool	What It Does
browser_navigate	Navigate to a URL
browser_snapshot	Capture accessibility snapshot of the current page
browser_take_screenshot	Take a PNG screenshot
browser_click	Click an element
browser_type	Type text into a field
browser_fill_form	Fill multiple form fields at once
browser_select_option	Select a dropdown value
browser_wait_for	Wait for text to appear or disappear
browser_find	Search the page for text or elements
browser_evaluate	Execute JavaScript on the page
browser_network_requests	List network requests made by the page
browser_press_key	Press a keyboard key
browser_hover	Hover over an element
browser_tabs	List, open, close, or switch browser tabs

Configuration

Configuration levels - Where does the server config live?

Before adding the server, decide which level suits your situation:

Level	File Location	When to Use
Global	~/.bob/mcp.json	Personal tools you want on every project on your machine
Project	.bob/mcp.json in your project root	Team integrations — commit to version control so every team member gets the same servers

> **Project-level takes precedence** over global when the same server name exists in both files.

Playwright → Use Global (`~/.bob/mcp.json`) Playwright is a personal browser automation tool with no project-specific settings. You want it available on every workspace on your machine.

How to open the config file in IBM Bob

1. Open VS Code with your project loaded
2. Click the **IBM Bob** icon in the Activity Bar
3. Click **Settings** (⚙️) → **MCP** tab
4. Click either button:
   - **Edit Global MCP**: opens `~/.bob/mcp.json` (all workspaces) ← *use this for Playwright*
   - **Edit Project MCP**: opens `.bob/mcp.json` in your project *(created automatically if missing)*
5. Add your server entry under `mcpServers` and **save**
6. Bob hot-reloads automatically — the server appears in the MCP tab within seconds

Setting environment variables for credentials

Reference credentials in `mcp.json` as `"${env:VAR_NAME}"` — never hardcode them.

Windows (PowerShell — persistent):

```
[System.Environment]::SetEnvironmentVariable('MY_API_KEY', 'your-value', 'User')
```

macOS / Linux:

```
echo 'export MY_API_KEY="your-value"' >> ~/.zshrc && source ~/.zshrc
```

> Restart VS Code after setting new environment variables so Bob picks them up.

Step 1: Opening the global config file

Follow the steps above and click **Edit Global MCP** to open `~/.bob/mcp.json`.

Step 2: Adding the configuration

Headless (default — no visible browser):

```
{
  "mcpServers": {
    "playwright": {
      "command": "npx",
      "args": ["-y", "@playwright/mcp@latest"]
    }
  }
}
```

Headed (visible browser window):

```
{
  "mcpServers": {
    "playwright": {
      "command": "npx",
      "args": ["-y", "@playwright/mcp@latest", "--headed"]
    }
  }
}
```

Step 3: Verifying

Settings → MCP tab → confirm `playwright` shows as connected.

Optional flags

Flag	Description
--headed	Show the browser window
--browser chromium	Use Chromium (default)
--browser firefox	Use Firefox
--browser webkit	Use WebKit (Safari engine)

> **Note:** These are optional configuration flags. Use `--headed` for visible browser interaction and `--browser` to select a specific browser engine based on your testing or compatibility requirements.

Example prompts for SAP projects

```
1. Take a screenshot of the SAP Fiori launchpad at https://my-sap.company.com.

2. Navigate to SAP Cloud ALM and list all open tasks in the current project.

3. Go to our Jira board, find Sprint 42, and take a screenshot of the backlog.

4. Open https://my-app.com/login, fill username 'testuser', click Login, and screenshot the result.

5. Check if the text 'Purchase Order Created' appears after submitting the form.
```

Auto-approve configuration

It will render cleanly as a JSON code block, with no straight separator lines:

```
{
  "mcpServers": {
    "playwright": {
      "command": "npx",
      "args": [
        "-y",
        "@playwright/mcp@latest",
        "--headed"
      ],
      "alwaysAllow": [
        "browser_navigate",
        "browser_take_screenshot",
        "browser_snapshot",
        "browser_click",
        "browser_type",
        "browser_wait_for",
        "browser_find"
      ]
    }
  }
}
```

Troubleshooting

Problem	Cause	Fix
Server not connecting	Node.js not on PATH	Run node --version to verify
Browser doesn't open in headed mode	No display available (remote server)	Use headed mode on local machines only
npx download hangs	Corporate proxy/firewall	npm config set proxy http://proxy:8080
Screenshot is blank	Page not fully loaded	Use browser_wait_for before screenshot
Tools not shown	Server connected but not loaded	Click Restart (🔄) in MCP tab

Complete Configuration Example

```
{
  "mcpServers": {
    "playwright": {
      "command": "npx",
      "args": ["-y", "@playwright/mcp@latest", "--headed", "--browser", "chromium"],
      "alwaysAllow": [
        "browser_navigate",
        "browser_take_screenshot",
        "browser_snapshot",
        "browser_click",
        "browser_type"
      ],
      "disabled": false
    }
  }
}
```

IBM Bob and SAP Business Technology Platform (SAP BTP)

Enterprises running SAP landscapes increasingly look to combine agentic AI tooling with the SAP Business Technology Platform (SAP BTP) to accelerate development, administration, and governance tasks. IBM Bob is one such tool. This section documents a supported, repeatable pattern for connecting IBM Bob to SAP BTP through the Model Context Protocol (MCP), using the MCP server for BTP Administration that SAP itself has released.

This section has three objectives:

- Establish a shared understanding of where SAP BTP sits within the broader SAP application landscape, so that the integration is applied to the correct deployment category.
- Explain the governing principle for any IBM Bob – SAP integration: prefer SAP-released tooling and protocols over community or custom alternatives.
- Provide a step-by-step, reproducible configuration procedure for both supported authentication modes, together with the security practices that should accompany them.

This section is limited to the integration between IBM Bob and SAP BTP through the BTP Administration MCP server. It does not cover integration with SAP's on-premise application layer (Type 1) or with SAP's software-as-a-service portfolio (Type 3), both of which are introduced in the SAP application landscape section for context but are addressed elsewhere in this publication.

The SAP application landscape: setting the context

Before examining the integration itself, it is necessary to be precise about what SAP BTP is and where it sits within a typical enterprise landscape. SAP organizes its application portfolio into three distinct deployment categories, and the correct integration strategy depends on which category is in scope.

Type 1

On-premise applications

These are SAP systems installed directly on infrastructure that the client controls, whether on physical on-premise hardware or a private hyperscaler tenancy on Microsoft Azure, Google Cloud Platform, or Amazon Web Services. SAP S/4HANA is the canonical example. The client organization owns the runtime environment, manages the transport lifecycle, and bears full responsibility for infrastructure operations.

Type 2

BTP applications

SAP Business Technology Platform is a subscription-based platform hosted on SAP's own managed hyperscaler infrastructure. Subscribers do not receive on-premise installation packages; instead, they access services — including Integration Suite, Build Code, and SAP Business Application Studio, among several thousand others — under a pay-as-you-go or monthly subscription model. The application runtime resides in SAP's environment, and the client connects to it over a secured channel from its own systems.

> **Note:** Because the BTP runtime is not hosted on client infrastructure, IBM Bob integration with BTP is necessarily API-driven rather than file-system-driven. Every capability described in this section is exposed through a programmatic interface, not through direct access to a file system or database.

Type 3

SAP SaaS offerings

A separate category covers SAP's proprietary software-as-a-service products, including SAP Ariba, SAP Concur, SAP SuccessFactors, SAP Sales Cloud (C4C), and SAP Commerce Cloud. These run on SAP-managed

infrastructure under subscription terms that are distinct from BTP. Integration with this category is outside the scope of this section.

Positioning IBM Bob against SAP Joule

SAP Joule — SAP's AI assistant family, which includes Joule for Developers and Joule for Consultants — is itself a BTP application. It runs inside SAP's cloud and is made available to subscribers as part of their BTP entitlements.

This positioning is important when comparing IBM Bob and SAP Joule for senior stakeholders evaluating tool strategy: both interact with the BTP ecosystem, but from fundamentally different vantage points.

SAP Joule	IBM Bob
Operates as a tenant inside BTP.	Operates as an external agent that connects to BTP through standard, published APIs.
Delivered and licensed as part of the BTP/SAP subscription entitlement.	Delivered independently of the BTP subscription; connects to BTP as one of potentially many integrated systems.

The integration principle: SAP-released tooling first

Any IBM Bob integration with SAP BTP should begin with the tools and protocols that SAP has released itself. This is a governance principle, not a matter of preference. SAP-released connectors carry SAP's own support commitment, follow SAP's versioning and deprecation policy, and reduce the risk of integration breakage across BTP release cycles.

SAP has released an MCP server for BTP Administration. This server exposes a headless API surface over the BTP platform, with endpoints for tenant management, subscription operations, and client identity services. It is the entry point for any IBM Bob integration that needs to read or act on BTP configuration programmatically.

> **Attention:** As of the time of writing, SAP has not published the BTP Administration MCP server as a publicly listed artifact. It was released targeting three specific clients — Build Code, Visual Studio Code, and Claude with Cline — rather than being announced as a general-purpose server. IBM Bob is not on that explicit list; however, the server itself is standard MCP-compliant, and the configuration approach described in this section has been verified to work with it. Project teams should monitor SAP's official MCP documentation for any change to the server's distribution or support status before taking this pattern into production.

Authentication options

Connecting IBM Bob to the SAP BTP Administration MCP server requires resolving an authentication decision first. Two approaches are available, and the correct choice depends on the client's security posture and BTP subscription configuration. Registering the MCP server for SAP BTP Administration in an AI client allows that client to manage the BTP account using natural-language prompts.

Prerequisites

Confirm the following prerequisites before beginning configuration:

- An MCP-compatible AI client is installed. SAP's published documentation lists Claude Code, GitHub Copilot in Visual Studio Code, and OpenCode as supported clients; this section documents the equivalent configuration for IBM Bob.
- Access to an SAP BTP global account is available.
- The applicable authentication mode has been determined, using the following table.

Authentication mode	When to use / what is needed
Single Sign-On (SSO)	Use when the BTP platform user is sourced from the Default Identity Provider (accounts.sap.com). Requires: an SAP user account and browser access for a one-time, interactive login flow.
Direct Connection (Basic Auth)	Use when the BTP platform user is sourced from a custom identity provider configured as a platform trust in the global account. Requires: the BTP username and password (Base64-encoded), and the IAS tenant subdomain.

Single Sign-On via browser delegation

This approach uses single sign-on through an external browser session. Rather than embedding credentials directly in the MCP configuration, the authentication flow delegates to the browser: the developer completes the BTP login interactively, and the MCP server picks up the resulting session.

This approach requires one additional component, the Playwright MCP server, which provides programmatic browser control that IBM Bob can invoke as a tool. With Playwright, the SSO handshake can finish without manual intervention beyond the initial, one-time session setup. Installing Playwright MCP alongside the BTP Administration MCP server is a configuration prerequisite for this path.

Basic authentication with Base64-encoded credentials

This approach uses the BTP subscription credentials directly. The developer takes the BTP user ID and password, encodes the pair in Base64, and supplies the encoded string as the authentication header in the MCP server configuration.

This approach is simpler to configure and does not require Playwright, but it embeds credentials in the configuration. Any project adopting it must ensure that the encoded credentials are stored in a secrets manager or environment variable, and never committed to version control alongside the `mcp.json` configuration file.

Configuring the BTP Administration MCP server in IBM Bob

This section provides the reproducible, step-by-step procedure for both authentication modes. It is the operational core of this section and is aimed primarily at SAP developers responsible for implementing the connection.

Step-by-step: Single Sign-On configuration

Step 1 — Install the Playwright MCP server. From the IBM Bob Marketplace, install "Playwright" MCP. This gives IBM Bob the ability to interact with a browser session. Confirm that the server shows as connected under Bob Settings → MCP, and that its browser-action tools are listed.

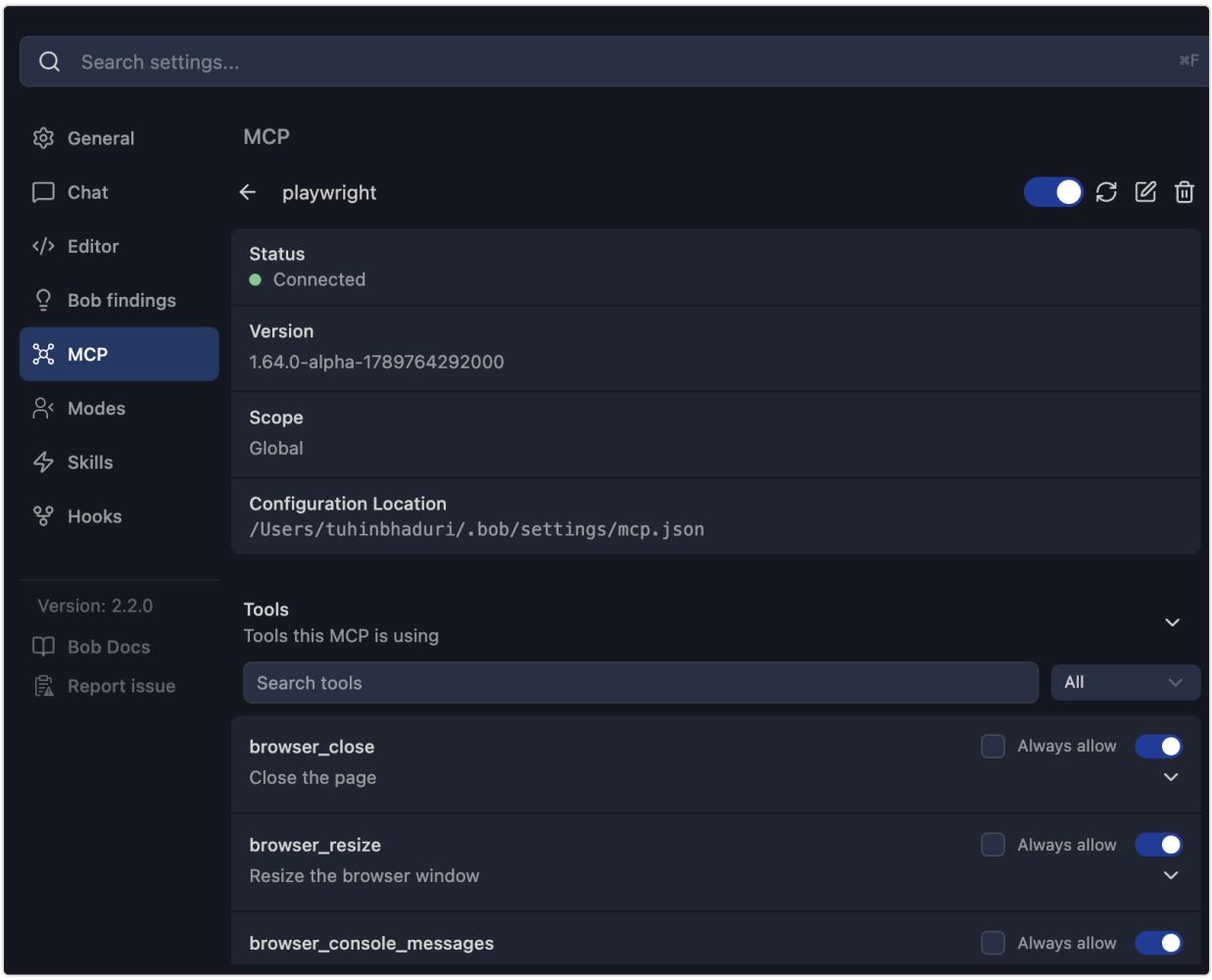


Figure: Playwright MCP server connected status in Bob Settings

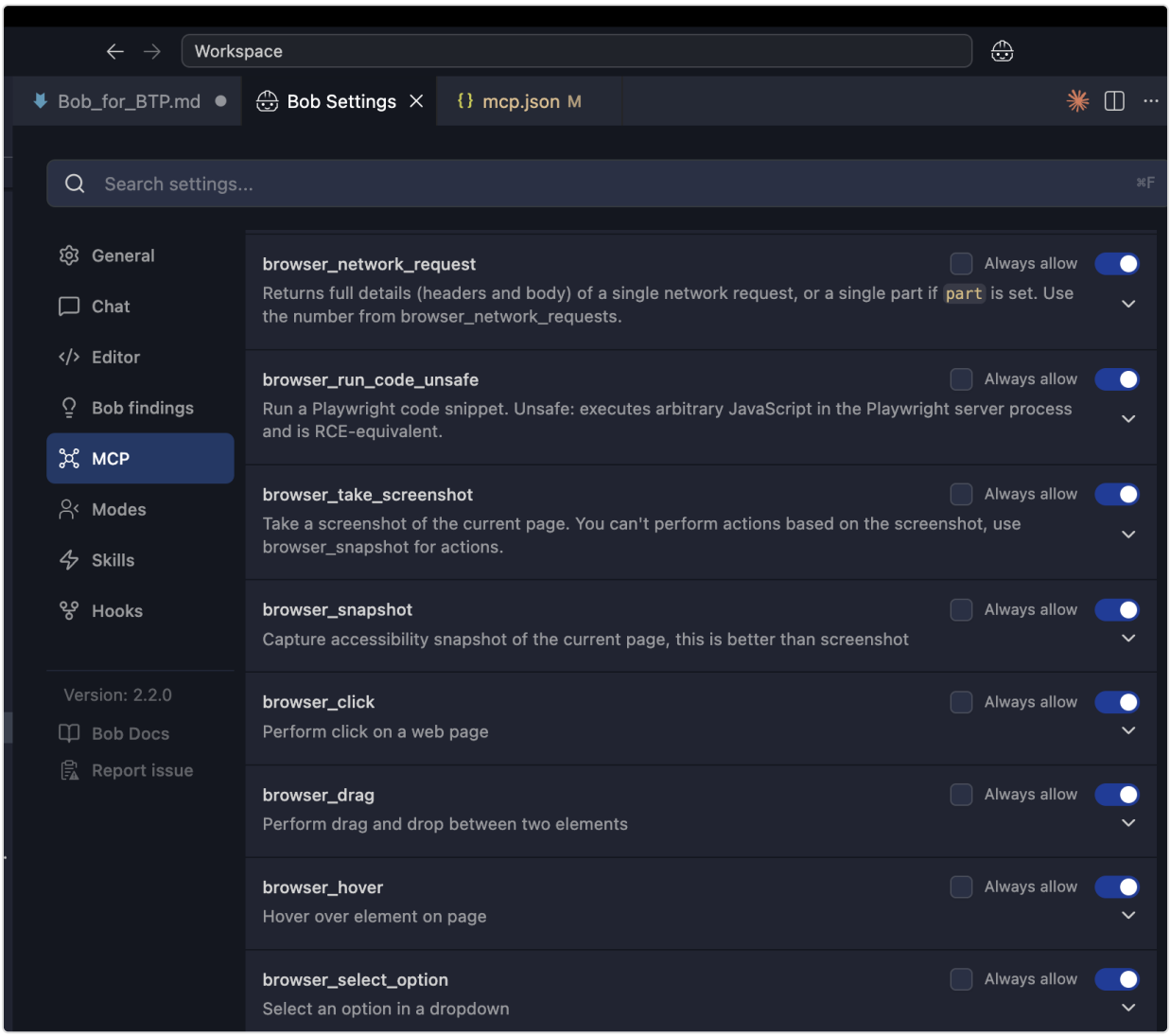


Figure: Playwright MCP browser-action tool set available to IBM Bob

Step 2 — Register the BTP Administration server. Open `mcp.json` under the `/.bob` folder and add the following entry, then save the file:

```
"BTP Administration": {
  "type": "streamable-http",
  "url": "https://sso.mcp.btp.cloud.sap/mcp",
  "oauth": {
    "clientId": "e789ba01-5612-47ee-bfe7-79e26411c1ca"
  }
}
```

Step 3 — Confirm registration and authenticate. Confirm that "BTP Administration" now appears under Settings → MCP. Return to the MCP settings screen in IBM Bob; an "Authenticate" button appears under "Status" for the BTP Administration server, alongside a warning that the server requires authentication — expected at this stage, since no authentication mechanism has yet been supplied in `mcp.json`.

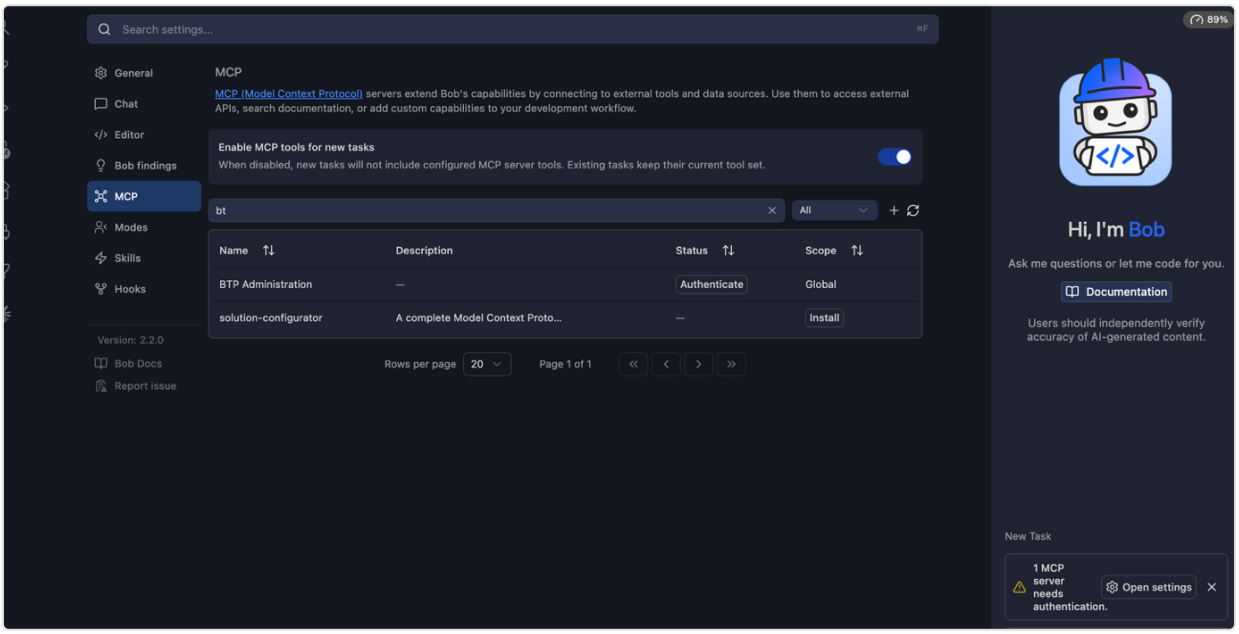


Figure: BTP Administration server registered but awaiting authentication

Ensure the Playwright MCP server is in connected mode before proceeding. Select Authenticate; a browser window opens and displays "Authentication Complete" once the SSO handshake finishes.

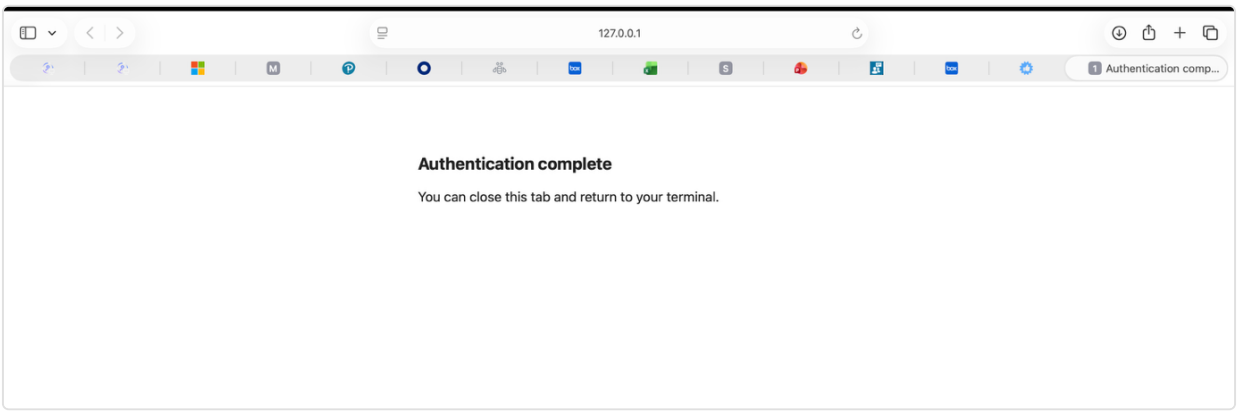


Figure: Browser confirmation of completed SSO authentication

Returning to the IBM Bob MCP settings screen, the BTP Administration server now shows a green connected indicator.

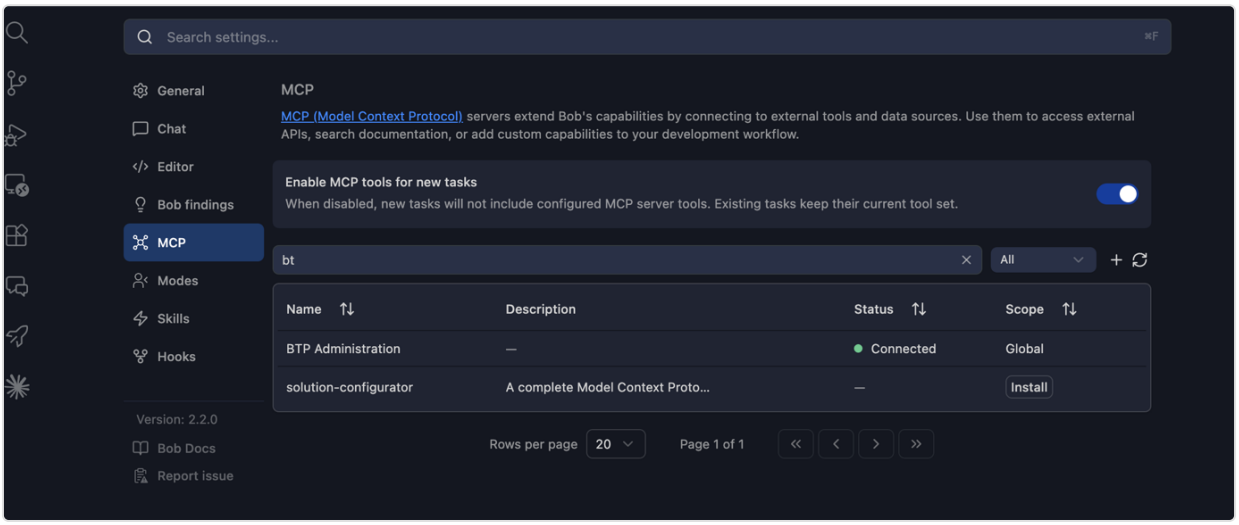


Figure: BTP Administration MCP server connected after successful SSO authentication

Step 4 — Verify the connection. With the BTP Administration server connected to the target BTP global account, submit a natural-language prompt in IBM Bob to confirm that the integration responds as expected.

Step-by-step: Basic authentication configuration

The configuration follows IBM Bob's standard MCP server registration pattern. The BTP Administration server is registered in the project's `mcp.json` file, alongside any other MCP servers the project uses. For the Base64 credential approach, register the server as shown in the following example, where `<base64-encoded-credentials>` is the encoded `user:password` string:

```
{
  "mcpServers": {
    "sap-btp-admin": {
      "command": "npx",
      "args": ["-y", "@sap/mcp-server-btp-administration"],
      "env": {
        "BTP_AUTH": "Basic <base64-encoded-credentials>",
        "BTP_ENDPOINT": "<your-btp-api-endpoint>"
      }
    }
  }
}
```

For the SSO approach, the Playwright MCP server must be registered first, and the BTP Administration server configured to use the resulting browser session for its authentication handshake, as described in the Single Sign-On configuration steps above. The exact configuration parameters depend on the target BTP subaccount and identity provider; consult the SAP BTP Administration MCP server documentation for current parameter names and endpoint patterns.

Security best practices for credential management

> **Warning:** Do not store Base64-encoded credentials directly in `mcp.json` if that file is version-controlled. Use environment-variable substitution or a secrets management tool, and reference the variable by

> name in the configuration. A leaked Base64-encoded credential is functionally equivalent to a leaked plaintext password — Base64 is an encoding, not encryption.

- Store encoded credentials in a secrets manager or environment variable, never in a committed configuration file.
- Rotate BTP service credentials on the same cadence used for other privileged administrative accounts.
- Where the security posture allows it, prefer SSO over Basic Authentication, because SSO avoids embedding long-lived credentials in any configuration artifact.
- Restrict the BTP account used for this integration to the minimum entitlements required for the intended administrative tasks.

Capabilities enabled after integration

With the BTP Administration MCP server configured and authenticated, IBM Bob gains a structured, tool-callable interface to the BTP platform. The scope of what is possible depends on the entitlements and API surface exposed by the specific BTP subaccount, but the following capabilities are representative of what the integration enables:

- Querying subaccount structure, entitlements, and service subscriptions.
- Reading environment configuration and service instance details.
- Supporting documentation and specification generation that references live BTP configuration, rather than a manually maintained inventory.

No custom RFC connector or proprietary adapter is required: IBM Bob gains programmatic access to BTP platform data as a set of standard, MCP-compliant tool calls.

Tool inventory overview

The BTP Administration MCP server exposes 61 tools organized into 9 capability groups, reachable over Streamable HTTP transport and secured with OAuth 2.0. This inventory represents the full administrative surface currently available to IBM Bob through the server, and is the reference point for scoping what any given automation or agent workflow can and cannot do.

Capability groups

The following table summarizes each capability group, its tool count, and representative tools within it. Tool names follow a consistent "Entity-Action" naming convention, described further in the following quick reference.

Table: Capability group tools listing

Capability group	Tools	Representative tools
Account Management	5	GlobalAccount-list / getSubaccount-list / get / create / update / deleteRegion-list
Entitlements & Quotas	7	GlobalAccountServiceEntitlement-listGlobalAccountServiceAssignment-listDirectoryServiceEntitlement/ Assignment-listSubaccountServiceQuota-listSubaccountEntitlement-assignDirectoryEntitlement-assign
Subscriptions	4	Subscription-list / getSubaccount-subscribeSubaccount-unsubscribe
Environments	6	Environment-list / getEnvironmentInstance-list / getCloudFoundryEnvironment-enable
User & Role Management	12	User-list / getRole-list / get / add / removeRoleCollection-list / create / getRoleCollection-assign / unassign / delete / updateRoleCollection-listUserAssignments
Trust & Identity	5	TrustConfiguration-list / create / getAvailableIDP-list / get
Service Instances & Bindings	9	ServiceInstance-list / get / create / update / deleteServiceBinding-list / getServicePlan-list / get
Destinations	8	Destination-list / get / create / update / deleteDestinationTrust-getDestinationCertificate-list / get
Navigation	1	Navigation-buildLink — generates direct BTP Cockpit URLs for any entity

Common use cases

The following end-to-end workflows illustrate how the capability groups above combine in practice. Each is representative of a task that would otherwise require multiple manual steps across the BTP Cockpit, the btp CLI, and service-specific dashboards.

Use case 1 — Onboard a new project subaccount end-to-end

Provision a new subaccount, assign entitlements, enable the Cloud Foundry environment, subscribe to applications, and set up role collections for the project team, all within a single conversation.

→ `Subaccount-create`, selecting a region with `Region-list`
→ `SubaccountEntitlement-assign`, to enable the required service plans
→ `CloudFoundryEnvironment-enable`, to create the Cloud Foundry org
→ `Subaccount-subscribe`, to subscribe to SAP Build Apps and other required applications
→ `RoleCollection-create`, `Role-add`, and `RoleCollection-assign`, to grant the project team access
→ `Navigation-buildLink`, to share the resulting Cockpit URL with the team

Use case 2 — Audit user access across subaccounts

Answer "who has administrative access to which subaccounts?" directly, without signing in to the Cockpit.

→ `Subaccount-list`, to retrieve all subaccounts in the global account
→ `RoleCollection-listUserAssignments`, to check each critical role collection
→ `User-get`, to verify individual user details and identity origin

Use case 3 — Manage destinations for connectivity

Create, update, or troubleshoot the destinations used by CAP applications, Integration Suite, or other BTP services.

→ `Destination-list`, to view existing destinations in a subaccount
→ `Destination-create`, to add a new HTTP or RFC destination with the appropriate authentication configuration
→ `DestinationTrust-get`, to retrieve trust certificates for mTLS setups
→ `DestinationCertificate-list`, to manage client certificates

Use case 4 — Entitlement capacity planning

Check remaining quota before provisioning new service instances, to avoid provisioning failures.

→ `GlobalAccountServiceEntitlement-list`, to review total quota and the remaining amount
→ `GlobalAccountServiceAssignment-list`, to see how quota is currently distributed
→ `SubaccountServiceQuota-list`, to confirm subaccount-level consumption

Use case 5 — Set up a custom Identity Provider (IdP) trust

Configure a corporate SAP Cloud Identity Services (IAS) tenant for single sign-on in a subaccount.

→ `AvailableIDP-list`, to discover available IAS tenants
→ `TrustConfiguration-create`, to establish trust with the IAS tenant
→ `TrustConfiguration-list`, to verify that the configuration is active

Operational best practices

> **Warning Caution — destructive operations:** `Subaccount-delete`, `RoleCollection-delete`, `Subaccount-unsubscribe`, and `ServiceInstance-delete` are irreversible or disruptive. Always confirm the target entity with a corresponding `-get` or `-list` call before invoking any of these tools.

- **Always list before acting**: use `-list` or `-get` tools to confirm the exact ID or name before any create, update, or delete operation. Identifiers are UUIDs and should never be guessed.
- **Use `Navigation-buildLink` to verify**: after any change, generate a Cockpit URL with this tool and visually confirm the result in the browser.
- **Check entitlements before provisioning**: run `GlobalAccountServiceEntitlement-list` before `ServiceInstance-create` or `SubaccountEntitlement-assign` to avoid quota-exhaustion errors.
- **Scope role assignments correctly**: pass `subaccount_id` or `directory_id` explicitly; without scoping, role operations default to the global account level, which is often not the intended target.
- **Use "enable" versus "amount" correctly for entitlements**: plans without a numeric quota require `enable: true`; plans with a numeric quota require `amount`. Using the wrong parameter causes the assignment to fail silently.
- **Check `blockingReason` on subaccount operations**: `Subaccount-create` and `Subaccount-get` return a `blockingReason` field; a subaccount still in the `CREATING` state cannot yet be used.
- **Delete bindings before instances**: service instances with active bindings cannot be deleted; call `ServiceBinding-list` and remove bindings first, then delete the instance.
- **Batch related operations in one session**: group logically related steps (for example, create subaccount → assign entitlements → enable Cloud Foundry) into a single conversation so that context is retained and errors are caught early.

Quick reference: tool naming convention

Tool names across all nine capability groups follow a consistent suffix convention that indicates both the action performed and its risk profile:

Table: Tool name quick reference

Suffix	Action	Risk profile
-list	List all entities of a type	Read-only
-get	Get details of a specific entity	Read-only
-create	Create a new entity	Irreversible
-update	Modify an existing entity	Verify first
-delete	Remove an entity permanently	Destructive
-assign	Link an entity to a user or subaccount	Verify scope
-unassign	Remove a link	Verify scope
-enable	Activate an environment or feature	One-time operation

Connecting other SAP-supported MCP clients

IBM Bob is one of several MCP-compatible clients that can connect to the BTP Administration MCP server. SAP publishes the authoritative, standard configuration guidance for its officially listed clients on the SAP Help Portal, and this guidance should be treated as the source of truth ahead of any client-specific instructions reproduced in this publication, including for IBM Bob itself:

SAP Help Portal — Connect to the MCP Server for SAP BTP Administration

At the time of writing, SAP's published guidance explicitly lists Claude Code, GitHub Copilot in Visual Studio Code, and OpenCode as supported clients, with the same MCP endpoint and OAuth client ID used throughout this section:

```
Endpoint:    https://sso.mcp.btp.cloud.sap/mcp
OAuth clientId: e789ba01-5612-47ee-bfe7-79e26411c1ca
```

Roadmap considerations

This is an early-stage integration relative to what SAP's MCP ecosystem is expected to offer over time. The BTP Administration server is a first step; SAP has signaled intent to expand the MCP surface across its BTP product portfolio. As additional servers become available — for example, covering Integration Suite operations, SAP Cloud Application Programming (CAP) model deployment, and BTP security administration — the same configuration pattern described in this section is expected to apply.

Architects and project managers should treat this as a standing item for periodic review rather than a one-time integration decision: as SAP's official MCP catalog grows, subsequent servers should be evaluated and adopted using the same governance principle set out in the integration principle section above.

Summary and key takeaways

SAP BTP is a subscription platform running on SAP-managed infrastructure, distinct from on-premise SAP systems (Type 1) and from SAP's SaaS portfolio (Type 3). IBM Bob integrates with it through the BTP Administration MCP server that SAP has released, using either SSO via Playwright or Base64-encoded Basic Authentication.

The practical steps are straightforward once the authentication approach is chosen: register the MCP server in `mcp.json`, handle credentials securely, and confirm the endpoint configuration against the target BTP subaccount. IBM Bob then gains programmatic access to BTP platform data as a set of standard tool calls.

Two principles should guide any team taking this integration into production:

- **Prefer SAP-released MCP servers over community alternatives** wherever they exist. The BTP Administration server is the right starting point precisely because it carries SAP's own release and support commitment.
- **Treat the expanding SAP MCP catalog as an evolving asset.** The integration pattern documented in this section is stable, but the breadth of what it covers will grow, and the roadmap should be revisited periodically.

> **Note:** Content reflects the BTP Administration MCP server as available at the time of writing. Readers should verify current server availability, distribution status, and parameter names against SAP's official MCP documentation before implementing this pattern in a production environment.

IBM Bob and ServiceNow: Integrating the AI Developer Assistant with Enterprise Change Management

ServiceNow is the system of record for enterprise change, incident, and release management in most large organizations running SAP landscapes. IBM Bob is an AI-powered developer assistant that works inside the software development lifecycle — it reads requirements, generates code, reviews artifacts, and closes the audit loop. Connecting the two is not a cosmetic integration. It is a structural one.

When Bob can read a change request directly before generating code, the code it produces reflects the actual requirement — not a developer's paraphrase of it. When Bob can write an observation back to a service ticket after a review, the audit trail closes automatically, without anyone having to remember to update the system.

This section covers the business case for the integration, the four categories of data that ServiceNow holds and why each matters to an autonomous SDLC, and the technical steps required to establish a secure, token-authenticated channel between Bob and a ServiceNow instance. The implementation uses the Model Context Protocol (MCP), OAuth 2.0 with JWT bearer authentication, and a Node.js server module that handles token acquisition, record retrieval, and field validation.

A working reference application —Demo Compliance Checker for Test Evidence, the IBM Test Evidence Review and Readiness Assistant — demonstrates the complete pattern against the `pm_project_task` and `change_request` tables. By the end of this section, a developer has everything needed to stand up the integration, and an architect has enough detail to evaluate it against the security and governance requirements of their organization.

ServiceNow as the Enterprise Data Foundation

Before examining the integration itself, it is worth being precise about what ServiceNow holds and why that data matters to IBM Bob. Most SAP program teams treat ServiceNow as a ticketing system. That framing undersells it considerably.

The four data categories

In practice, ServiceNow functions as a structured repository for four distinct categories of operational data, each of which is directly relevant to an autonomous development workflow.

Change requests

A change request — identified by a `CHG` prefix — captures the formal approval to promote an enhancement or correction to production. It contains the business justification, implementation plan, risk and impact analysis, and the assigned owner. Every production deployment in a governed SAP landscape is traceable to a change request. In change-driven delivery models, the change request is the requirement specification.

Production incidents

An incident record describes a failure or degradation in a live system. When Bob assists with a defect fix, having direct read access to the incident record — rather than a paraphrase passed through a developer — reduces the chance that a critical detail is lost in translation. The difference between "slow performance on the goods receipt screen" and the precise error code, affected system, and business impact can materially change how the fix is approached.

Service requests and change tasks

These are sub-units of work raised either by end users or decomposed from a larger change. The `CSPRJTASK` prefix identifies the project task variant used in SAP program management contexts. These records carry the granular work scope that maps most directly to the code a developer writes. A task record that says "configure payment terms for vendor group X" is specific enough for Bob to generate a targeted ABAP configuration report or BDC script.

Agile delivery artifacts

ServiceNow's Agile Development module stores user stories, sprint plans, and backlogs alongside the change and incident data. This means a single platform can serve as the source of truth from business requirement through to production approval — a span that typically crosses four or five disconnected tools in organizations that have not consolidated onto ServiceNow.

Why centralization matters

The argument for connecting Bob to ServiceNow rests on a single observation: the information Bob needs to do its job well already exists inside ServiceNow. Organizations that store requirements in one system, change approvals in another, and test evidence in a third force their developers to gather context manually before they can act. That gathering takes time, and it introduces transcription errors.

ServiceNow's value as an integration target is precisely its breadth. A single authenticated connection gives Bob access to requirements, change approvals, incident context, and task scope — the full set of inputs that govern what a developer should build and how it should be deployed. The integration described in this section is designed to make that access programmatic, auditable, and secure.

IBM Bob's role in the SAP software delivery lifecycle

IBM Bob enters the delivery lifecycle at every stage where a developer or architect needs to produce, review, or transform artifacts. That scope is wider than it first appears.

Requirement analysis at sprint start

When a sprint begins, the developer's first task is to understand what the change request or user story actually requires. Bob can retrieve the change request directly from ServiceNow, parse the description and justification fields, and produce a structured requirement summary. The developer does not transcribe the ticket — Bob reads it. For a senior developer who handles five or six tickets a sprint, this alone saves a meaningful amount of preparation time each week.

Code generation and refactoring

When Bob generates ABAP, CAP, or Node.js code, the quality of that output depends on the quality of the context it receives. A change request number is sufficient context for Bob to pull the full record and use its fields — short description, implementation plan, risk analysis — as grounding for what it produces. Refactoring follows the same logic: the incident record describing the defect shapes the fix, and the change task describing the scope constrains what Bob is allowed to change.

Release note validation — The Demo Compliance Checker for Test Evidence pattern

A worked example of this pattern is the Demo Compliance Checker for Test Evidence application, described in detail in the Reference application: Demo Compliance Checker for Test Evidence and the AI-driven release note checker section. That application parses a `.docx` release note, extracts all `Change Task` and `CHG` numbers, queries ServiceNow for each, and validates that mandatory fields are populated. The validation result is returned to the reviewer before the release note reaches the change advisory board. This turns a manual, error-prone checklist exercise into an automated, auditable process.

Post-deployment feedback

Bob can write back to ServiceNow as well as read from it. After a deployment, it can update the change request state, attach a validation summary, or create a follow-on task if post-deployment checks identify an issue. Closing the loop programmatically is what separates a connected workflow from a fragmented one — and it is what gives the organization the audit evidence it needs to demonstrate governance compliance.

Integration architecture and authentication

The integration between IBM Bob and ServiceNow relies on three components: an OAuth 2.0 / JWT application registered in ServiceNow, a Node.js server module that handles token acquisition and REST API calls, and an MCP server entry in Bob's configuration that makes the integration available as a tool.

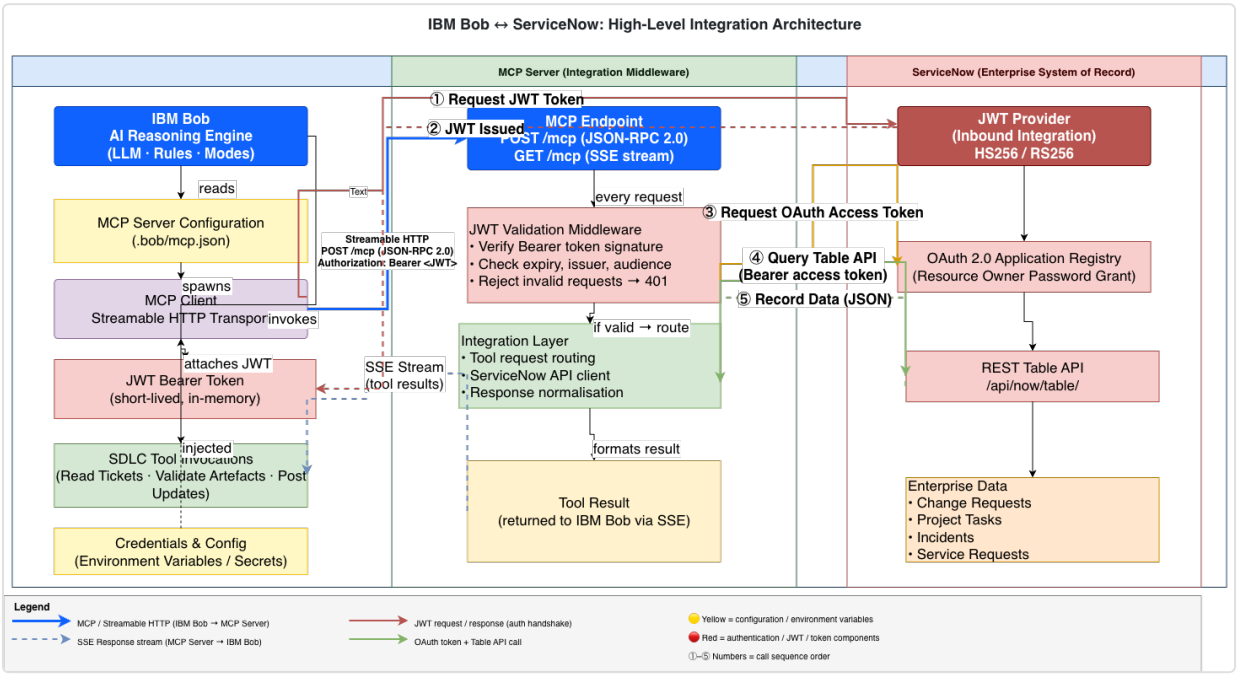


Figure: End-to-end architecture from IBM Bob through an MCP server to ServiceNow

The diagram shows three logical zones: the IBM Bob developer workstation on the left, the Node.js MCP server in the center, and the ServiceNow instance on the right. All communication flows left-to-right. Bob sends Streamable HTTP requests to the MCP endpoint carrying a short-lived JWT in the Authorization header. The MCP server validates that JWT, routes the tool call to the appropriate Express handler, which in turn authenticates against ServiceNow using the OAuth password grant and calls the Table API. The response flows back through the same chain.

Authentication model: JWT and OAuth 2.0

ServiceNow supports several OAuth 2.0 grant types. This implementation uses the JWT bearer grant for MCP-layer authentication and the Resource Owner Password Credentials grant for ServiceNow Table API calls. Understanding the distinction between the two is important for anyone who needs to review the security model.

The JWT bearer grant secures the channel between IBM Bob and the MCP server. Every request Bob sends carries an `Authorization: Bearer <JWT>` header. The MCP middleware extracts that token, verifies its signature using `HS256`

or `RS256`, and checks the issuer, expiry, and audience claims. Requests that fail validation are rejected with a 401 before any ServiceNow call is made. This ensures that only Bob — not any other process on the network — can invoke the MCP tools.

The Resource Owner Password Credentials grant handles the MCP server's authentication to ServiceNow. The server exchanges its credentials (client ID, client secret, username, password) for a short-lived Bearer token at the ServiceNow `/oauth_token.do` endpoint. That token is used on every subsequent Table API call and expires after 30 minutes.

> **Important:** The Resource Owner Password Credentials grant must be explicitly enabled in ServiceNow. An administrator must configure an OAuth Application Registry entry with the password grant type checked. Without that setting, the token endpoint returns an `access_denied` error regardless of whether the credentials are correct.

For organizations that require stronger authentication at the ServiceNow layer, the JWT Inbound Integration framework described in the Configuring ServiceNow: JWT Inbound Integration section is the recommended path for production deployments.

Configuring ServiceNow: JWT inbound integration

Before the MCP server can authenticate, a JWT Inbound Integration entry must exist in ServiceNow. The configuration is found under **System OAuth > Inbound Integrations** in the ServiceNow navigator. Searching for "oauth" in the navigator filter brings it up immediately.

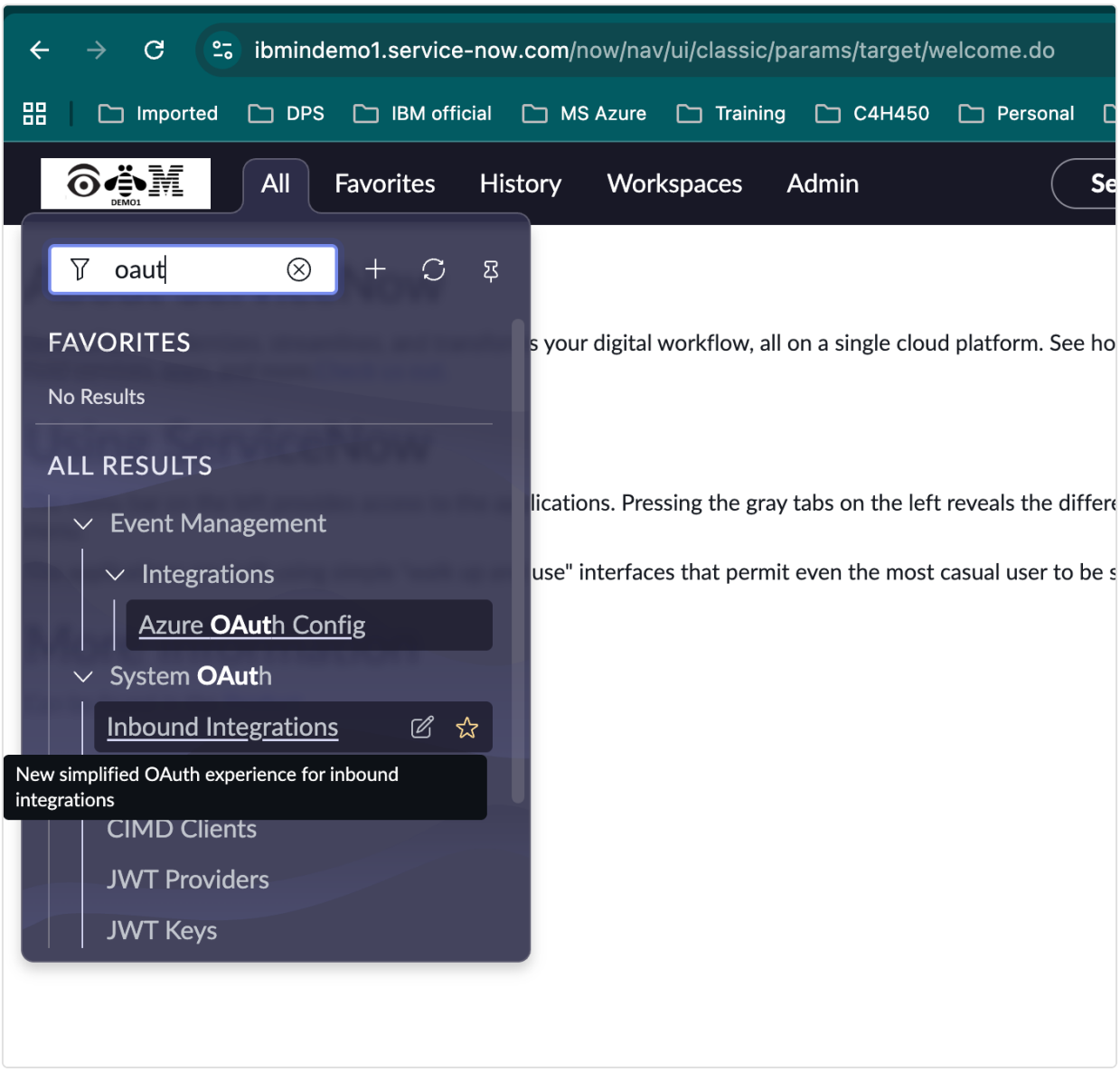


Figure: ServiceNow navigator search for oauth showing JWT settings

Once inside the Inbound Integrations module, create a new JWT bearer grant entry. The following form is the JWTMCP entry used in the reference implementation. The key fields are the Name (JWTMCP), Provider name (ServiceNow), Client ID, Client secret, and the User field mapping (Email). JTI verification should be enabled in any production configuration.

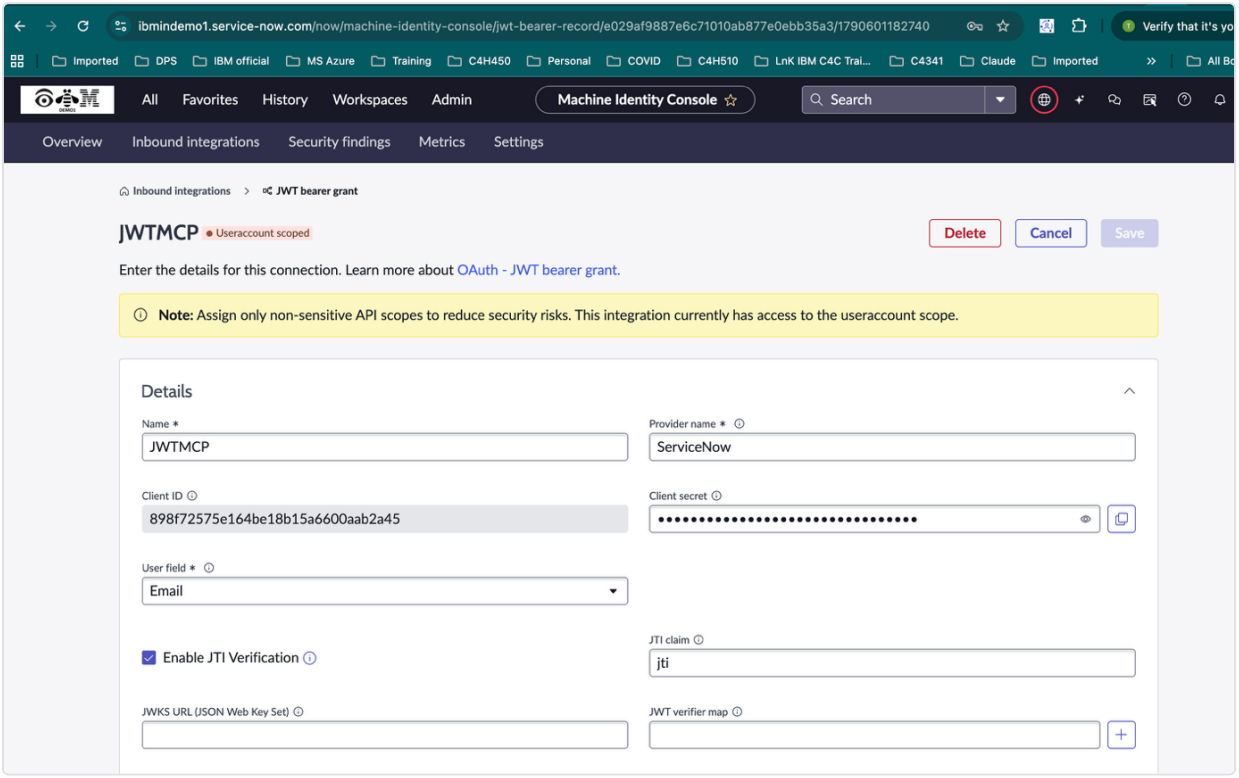


Figure: ServiceNow JWTMCP inbound integration configuration

The Auth Scope section of the same record controls which ServiceNow APIs this integration can access. The `userAccount` scope grants access to all resources available to the signed-in user, which is convenient but broad. For production deployments, create a named auth scope that limits the integration to the specific tables and operations the MCP server requires — typically read access to `change_request` and `pm_project_task`.

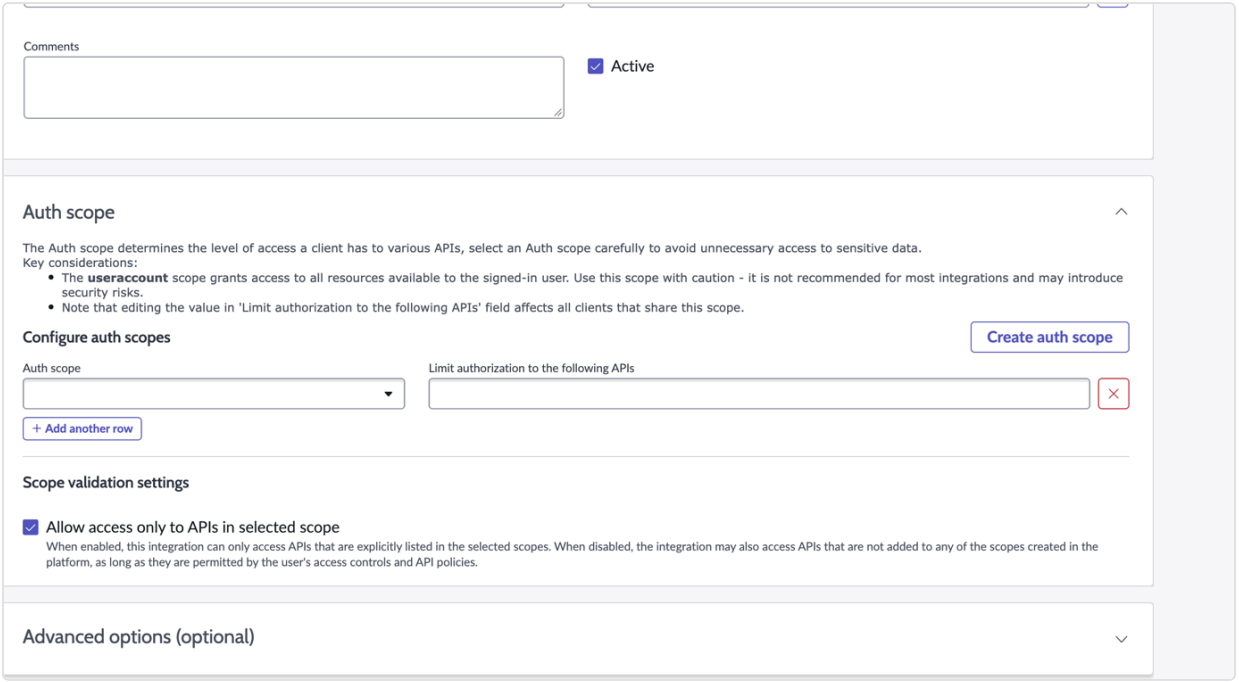


Figure: ServiceNow auth scope configuration for the JWTMCP integration

Environment configuration

All credentials and connection parameters are read from environment variables. Nothing is written into application source code or container images. The full set of required variables is:

```
# ServiceNow instance base URL — no trailing slash
SN_INSTANCE=https://your-instance.service-now.com

# OAuth application credentials registered in ServiceNow
SN_CLIENT_ID=<your-oauth-client-id>
SN_CLIENT_SECRET=<your-oauth-client-secret>

# ServiceNow user account used for the password grant
SN_USERNAME=<your-username>
SN_PASSWORD=<your-password>

# Optional overrides
SN_TABLE=pm_project_task    # default table for CSPRJTASK records
SN_TIMEOUT_MS=15000         # HTTP timeout in milliseconds
PORT=3000
NODE_ENV=development
```

> **Tip:** For local development, store these values in `release-note-checker-node/.env`. For production on IBM Cloud Code Engine, inject them as Code Engine secrets. They must never appear in source control or container images.

The server module

The `servicenow.js` module in `src/integrations/` isolates all ServiceNow API calls in one place. It exposes three functions: `getMissingConfig()`, which validates that all required environment variables are present before any network call is made; `getAccessToken()`, which performs the OAuth password grant and returns the Bearer token; and `fetchRecord()`, which queries the appropriate table for a given ticket number and returns a normalized record object.

Token acquisition:

```
// src/integrations/servicenow.js — OAuth password grant

async function getAccessToken() {
  const cfg = getConfig();
  const body = new URLSearchParams({
    grant_type:    'password',
    client_id:     cfg.clientId,
    client_secret: cfg.clientSecret,
    username:      cfg.username,
    password:      cfg.password,
  });
  const response = await fetch(`${cfg.instance}/oauth_token.do`, {
    method:  'POST',
    headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
    body:    body.toString(),
  });
  const data = await response.json();
  if (!data.access_token) {
    throw new Error(data.error_description || 'No access_token in ServiceNow response');
  }
  return data.access_token;
}
```

Record retrieval routes the query to the correct table based on the ticket number prefix — `change_request` for `CHG` numbers, and the configured task table for all others:

```
// src/integrations/servicenow.js — table routing and record fetch

function resolveTable(number) {
  return number.startsWith('CHG') ? 'change_request' : TASK_TABLE;
}

async function fetchRecord(number, token) {
  const cfg   = getConfig();
  const table = resolveTable(number);
  const params = new URLSearchParams({
    sysparm_query:         `number=${number}`,
    sysparm_fields:        'number,short_description,description,state,assigned_to,percent_complete',
    sysparm_limit:         '1',
    sysparm_display_value: 'true',
  });
  const response = await fetch(
    `${cfg.instance}/api/now/table/${table}?${params}`,
    { headers: { Authorization: `Bearer ${token}`, Accept: 'application/json' } }
  );
  const data = await response.json();
  return data.result?.[0] ?? null;
}
```

MCP server configuration in IBM Bob

IBM Bob connects to the ServiceNow integration through an MCP server entry in `.bob/mcp.json`. The entry launches the Node.js application as a local process and injects the ServiceNow credentials as environment variables. Bob hot-reloads this file on save, so the server starts immediately and its tools become available without a restart.

```
"release-note-checker": {
  "command": "npm",
  "args": ["start"],
  "cwd": "/path/to/release-note-checker-node",
  "env": {
    "SN_INSTANCE":      "https://your-instance.service-now.com",
    "SN_CLIENT_ID":     "<your-client-id>",
    "SN_CLIENT_SECRET": "<your-client-secret>",
    "SN_USERNAME":      "<your-username>",
    "SN_PASSWORD":      "<your-password>",
    "SN_TABLE":         "pm_project_task",
    "SN_TIMEOUT_MS":    "15000",
    "PORT":             "3000",
    "NODE_ENV":         "development"
  },
  "disabled": false
}
```

> **Tip:** Name the MCP entry after the integration's function rather than its technology stack. `release-note-checker` is more discoverable in the MCP panel than `servicenow-rest-client`.

Reference application: Demo Compliance Checker for Test Evidence and the AI-driven release note checker

- Demo Compliance Checker for Test Evidence— IBM Test Evidence Review and Readiness Assistant — is a demo Node.js application that demonstrates the ServiceNow integration pattern against a real-world use case: validating a SAP release note before it reaches the change advisory board. The application is the reference implementation for the patterns described in this section.

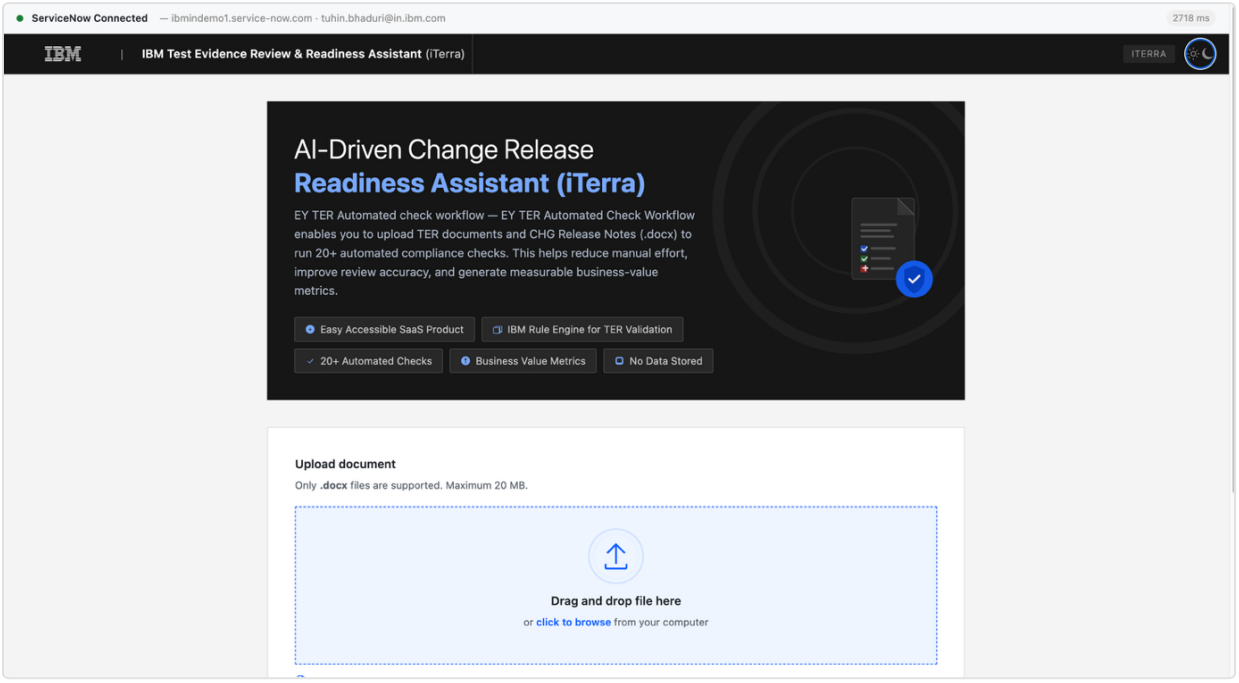


Figure: iTerra change release readiness assistant connected to ServiceNow

Notice the "ServiceNow Connected" indicator at the top-left of the screen, showing the live instance URL, the authenticated user, and the current API response latency (2,718 ms in this screenshot). This status is confirmed on every page load by a lightweight probe against the `pm_project_task` table. Teams know before they upload a document whether the ServiceNow connection is healthy.

What it does

The application accepts a `.docx` release note as a document upload, extracts all `CSPRJTASK` and `CHG` ticket numbers from the document text, queries ServiceNow for each, and returns a field-level compliance report. The key value proposition — spelled out on the landing screen — is straightforward: 20+ automated compliance checks, IBM rule engine for TER validation, measurable business-value metrics, and no data stored between sessions.

For change request records, the validator checks that the `description`, `justification`, `implementation_plan`, and `risk_impact_analysis` fields are populated. For project task records, it checks `description`, `percent_complete`, and `assigned_to`. Every check is surfaced as a per-ticket card showing the ticket's state, short description, and a table of field checks with clear OK or Blank status for every mandatory field.

Application structure

```
release-note-checker-node/

├── src/
│   ├── server.js                # Express application — HTTP routes
│   ├── integrations/
│   │   └── servicenow.js        # OAuth token acquisition and REST calls
│   ├── parsers/
│   │   └── docxParser.js        # .docx to plain text; regex extraction
│   ├── analyzers/
│   │   └── observations.js      # Field validation rules
│   └── public/
│       └── index.html           # Single-page web UI
├── .env.example                 # Environment variable template
├── Dockerfile                   # Node 20 Alpine image
└── package.json
```

API endpoints

Table: Demo Compliance Checker for Test Evidence **HTTP API endpoints**

Method	Path	Purpose
GET	/	Serves the single-page web UI
POST	/check	Runs 20+ document compliance checks against the uploaded .docx
POST	/analyze	Extracts ticket numbers and queries ServiceNow for each
POST	/ai-insight	Generates AI-assisted remediation advice via IBM watsonx.ai
GET	/api/sn-status	Returns ServiceNow connectivity status and latency

The `/api/sn-status` endpoint is particularly useful during deployment. It acquires a token and performs a lightweight probe against the `pm_project_task` table, returning a JSON payload containing connected status, instance URL, response latency, authenticated user, and REST API availability. The web UI calls this endpoint on page load and renders the result as a persistent connectivity banner — the green "ServiceNow Connected" badge shown above.

Running the application locally

After cloning the repository, copy `.env.example` to `.env`, populate the ServiceNow credentials, and start the server:

```
cd release-note-checker-node
npm install
cp .env.example .env
# Edit .env with your ServiceNow instance credentials
npm start
# Application available at http://localhost:3000
```

> **Note:** The server prints a confirmation line on startup and is ready to accept requests immediately. No database or additional infrastructure is required. The application is stateless — no data is persisted between sessions, as indicated on the landing page.

Design principles and architectural guidance

Several decisions in this implementation reflect broader principles that architects should carry into any IBM Bob–ServiceNow integration, regardless of the specific use case. These are not abstract guidelines — each one addresses a failure mode that teams encounter in practice.

Credentials never enter the source code

The separation between code and configuration is absolute in this design. Every credential — OAuth client ID, client secret, username, password — lives in an environment variable. The application reads it at runtime. This means the same Docker image can connect to a development instance, a quality assurance instance, or a production instance simply by changing the environment configuration. It also means that rotating a credential requires no code change and no redeployment of the application binary. Teams that embed credentials in source files inevitably face a rotation event that touches ten places and still leaves secrets in the git history.

Token acquisition is per-request

The current implementation acquires a fresh OAuth token on every call to `getAccessToken()`. Tokens expire after 30 minutes; acquiring one per request eliminates the need for a token cache and the associated expiry management logic. The cost is one additional HTTP round-trip per operation. At the call volumes typical of a developer assistant integration, that cost is negligible — typically 50–150 ms against a well-connected instance.

For high-throughput scenarios, a simple in-memory cache with a 25-minute TTL is the natural optimization. The module boundary in `servicenow.js` makes that change local — no other part of the application needs to know that caching was added. Do not optimize for this case until measurements show it is necessary.

Table routing by ticket prefix

ServiceNow's data model stores change requests and project tasks in separate tables with different field schemas. The routing logic in `resolveTable()` — a single conditional on the ticket number prefix — keeps the rest of the codebase free of that distinction. A caller passes a ticket number; the module determines the correct table and field set. Adding support for additional record types (incidents, user stories, problem records) requires only extending the routing function and the relevant field list, not restructuring the caller.

This pattern also makes the integration resilient to future schema changes. If the organization introduces a new task prefix, one line changes in `resolveTable()` and the rest of the system is unaffected.

The MCP boundary

Wrapping the integration as an MCP server creates a clean boundary between Bob's reasoning layer and the ServiceNow API. Bob invokes tools; it does not make HTTP calls directly. This means the authentication logic, error handling, field normalization, and table routing all live in one place, outside of any prompt or rule file. When the ServiceNow API changes — a field is renamed, a new table is introduced, authentication policy is updated — the change is made in the MCP server, not distributed across a collection of prompts that may not be immediately apparent to the person making the fix.

The MCP boundary also makes the integration testable in isolation. The Node.js server can be started independently, called with test ticket numbers, and its output inspected before Bob is ever involved. Teams that skip this boundary tend to find ServiceNow-related issues buried in Bob's reasoning logs rather than surfaced cleanly in a dedicated service.

Summary and key takeaways

ServiceNow holds the data that governs what an SAP developer should build, how a change should be deployed, and whether a release is compliant with the organization's governance policy. IBM Bob is most effective when it can read

that data directly rather than receiving paraphrased summaries through a developer. The integration described in this section establishes that direct connection.

The implementation rests on three mechanisms: JWT-based authentication against the ServiceNow instance, a Node.js server module that handles token acquisition and record retrieval, and an MCP server entry in Bob's configuration that makes the integration available as a callable tool. The Demo Compliance Checker for Test Evidence application demonstrates the pattern concretely — parsing a release note, querying ServiceNow for every referenced ticket, and returning a field-level compliance report before the document reaches the change advisory board.

Table: Key principles for IBM Bob–ServiceNow integration implementations

Principle	Guidance
Credentials management	Credentials belong in environment variables. Never in source code, never in container images, never in prompt files.
Token lifecycle	Acquire a fresh token per request unless call volume measurements justify a cache. A 25-minute TTL cache is the standard optimization when needed.
Table routing	Keep table and field resolution inside the integration module. Callers pass ticket numbers; the module handles the rest.
MCP boundary	Authentication, error handling, and field normalization belong in the MCP server, not in prompts or rules. This makes the integration independently testable and maintainable.
Scope discipline	Configure the ServiceNow auth scope to cover only the tables and operations the integration actually uses. Avoid userAccount scope in production.

The next section examines how IBM Bob's governance framework — rules, modes, and skill files — can be used to enforce the ServiceNow workflow consistently across a development team, ensuring that every developer connects to the same instance and follows the same field-validation policy without manual coordination.

Jira MCP — Agile Project Integration

This section covers connecting IBM Bob to Jira for agile issue tracking and sprint management.

Why Jira MCP?

Why would an AI assistant need access to Jira?

Imagine a project team preparing for a release. A team member asks IBM Bob:

"Show me all high-priority open issues for the current sprint, identify the blocked items, and summarize what needs attention."

Without direct access to Jira, Bob would need the user to manually provide the relevant issue information. With **Jira MCP**, Bob can interact with Jira through MCP tools and retrieve the information required to answer the request.

The Jira MCP server can expose capabilities such as searching issues, retrieving issue details, creating or updating issues, querying sprint information, and checking issue status.

The interaction can be summarized as:

```
User Request
     ↓
   IBM Bob
     ↓
   Jira MCP
     ↓
     Jira
     ↓
Issues / Sprint / Status Information
     ↓
   IBM Bob
     ↓
Analysis & Summary
     ↓
User Response
```

Key takeaways

With Jira MCP, an AI assistant can interact with live Jira project data and capabilities; users can search, analyze, and manage work items through natural-language requests.

Overview

Jira is the most widely used agile project management tool on SAP delivery engagements. The **Jira MCP server** connects IBM Bob directly to Jira, enabling Bob to create issues, query sprints, update statuses, and generate reports — all through natural language.

The following list describes possible SAP delivery use cases:

- Create Jira defects directly from Bob while reviewing code or test results
- Query open tickets for a sprint without leaving VS Code
- Generate sprint summary reports from Jira data
- Update issue status as part of a deployment workflow

Prerequisites

Requirement	Details
IBM Bob IDE	Installed and running
Node.js	v18 or later
Jira Account	Cloud or Server/Data Center
Jira API Token	Generated from your Atlassian account settings

Generating a Jira API token (Cloud)

1. Go to https://id.atlassian.com/manage-profile/security/api-tokens
2. Click **Create API token**
3. Give it a name (for example, `IBM-Bob-MCP`) and click **Create**
4. Copy the token — you will not be able to view it again
5. Set it as an environment variable:

Finding the Jira MCP server

Option A

Bob Marketplace

1. Settings (⚙️) → MCP tab → search **"Jira"** in the marketplace
2. Install if available

Option B

Community Packages

If not in the Bob marketplace, use a community package:

```
# Check availability
npx -y @modelcontextprotocol/server-jira --help
```

Popular community options on npm / GitHub:

- `mcp-server-jira`
- `@sooperset/mcp-atlassian`

Configuration

Configuration levels - Where does the server config live?

Before adding the server, decide which level is right for your situation:

Level	File Location	When to Use
Global	~/.bob/mcp.json	Personal tools you want available on every project on your machine
Project	.bob/mcp.json in your project root	Team integrations tied to this engagement — commit to version control so every team member gets the same servers automatically

> **Note:** Project-level takes precedence over global when the same server name exists in both files.

How to open the config file in IBM Bob (Step by step)

1. Open VS Code with your project loaded
2. Click the **IBM Bob** icon in the Activity Bar to open the Bob panel
3. Click the **Settings** icon (⚙️) at the top of the Bob panel
4. Select the **MCP** tab
5. You will see two buttons near the top of the tab:
   - **Edit Global MCP** — opens `~/.bob/mcp.json` *(applies to all your workspaces)*
   - **Edit Project MCP** — opens `.bob/mcp.json` inside your current project folder *(Bob creates the file automatically if it does not exist yet)*
6. Click the appropriate button — the JSON file opens in VS Code

7. Add your server entry under the `mcpServers` object and **save the file**
8. Bob detects the change and **hot-reloads automatically** — no restart needed
9. Switch back to the **MCP** tab — your new server appears in the list within seconds

Setting environment variables for credentials

All credentials must be stored as environment variables — never hardcoded in the JSON file. Reference them as `"${env:VAR_NAME}"` in `mcp.json`.

Windows (PowerShell — persistent):

```
[System.Environment]::SetEnvironmentVariable('MY_API_KEY', 'your-value-here', 'User')
```

macOS / Linux (persistent — add to shell profile):

```
echo 'export MY_API_KEY="your-value-here"' >> ~/.zshrc && source ~/.zshrc
```

> **Important:** After setting a new environment variable, restart VS Code so Bob picks up the new value.
> **Jira → Use Project** (`.bob/mcp.json`) Jira credentials and project URLs are specific to this engagement. Commit `.bob/mcp.json` to version control so every team member gets the integration automatically. Store the API token as an environment variable — never hardcoded.

Using `@sooperset/mcp-atlassian` (recommended community server)

```
{
  "mcpServers": {
    "jira": {
      "command": "npx",
      "args": ["-y", "@sooperset/mcp-atlassian"],
      "env": {
        "JIRA_URL": "https://your-org.atlassian.net",
        "JIRA_USERNAME": "your.email@company.com",
        "JIRA_API_TOKEN": "${env:JIRA_API_TOKEN}"
      }
    }
  }
}
```

For Jira server / data center (PAT authentication)

```
{
  "mcpServers": {
    "jira": {
      "command": "npx",
      "args": ["-y", "@sooperset/mcp-atlassian"],
      "env": {
        "JIRA_URL": "https://jira.yourcompany.com",
        "JIRA_PERSONAL_TOKEN": "${env:JIRA_PERSONAL_TOKEN}"
      }
    }
  }
}
```

Key tools exposed

Tool	What It Does
jira_get_issue	Retrieve full details of a specific issue
jira_search_issues	Search using JQL (Jira Query Language)
jira_create_issue	Create a new issue / defect / story
jira_update_issue	Update fields on an existing issue
jira_add_comment	Add a comment to an issue
jira_get_project	Retrieve project details
jira_list_projects	List all accessible projects
jira_get_sprint	Get active/closed sprint details
jira_get_board	List boards for a project
jira_transition_issue	Move an issue to a different status

Example prompts for SAP projects

```
1. "Show me all open defects in the SAP S4 Migration project for Sprint 12"
2. "Create a Jira bug: title 'MM60 report shows wrong totals after transport', project SAP-MAINT, priority
3. "How many story points are completed in the current sprint?"
4. "Add a comment to PROJ-1234: Confirmed fix deployed to QA on 30 Sep 2026"
5. "Move issue PROJ-456 to Done"
6. "Give me a summary of all Critical priority bugs still open in this project"
```

Auto-approve configuration

```
{
  "mcpServers": {
    "jira": {
      "command": "npx",
      "args": ["-y", "@sooperset/mcp-atlassian"],
      "env": {
        "JIRA_URL": "https://your-org.atlassian.net",
        "JIRA_USERNAME": "your.email@company.com",
        "JIRA_API_TOKEN": "${env:JIRA_API_TOKEN}"
      },
      "alwaysAllow": [
        "jira_get_issue",
        "jira_search_issues",
        "jira_get_sprint",
        "jira_list_projects"
      ]
    }
  }
}
```

> **Tip:** Only auto-approve read-only tools. Keep write tools (create/update/transition) on manual approval to avoid accidental changes.

Troubleshooting

Problem	Cause	Fix
401 Unauthorized	Wrong credentials	Re-check JIRA_USERNAME and JIRA_API_TOKEN
403 Forbidden	Missing project permissions	Check Jira project role for your account
"Project not found"	Wrong project key	Use jira_list_projects to find the correct key
JQL errors	Invalid query syntax	Test JQL in Jira's issue navigator first
Server not connecting	Node.js missing	Verify node --version

Complete configuration example

```
{
  "mcpServers": {
    "jira": {
      "command": "npx",
      "args": ["-y", "@sooperset/mcp-atlassian"],
      "env": {
        "CONFLUENCE_URL": "https://your-org.atlassian.net/wiki",
        "CONFLUENCE_USERNAME": "your.email@company.com",
        "CONFLUENCE_API_TOKEN": "${env:JIRA_API_TOKEN}",
        "JIRA_URL": "https://your-org.atlassian.net",
        "JIRA_USERNAME": "your.email@company.com",
        "JIRA_API_TOKEN": "${env:JIRA_API_TOKEN}"
      },
      "alwaysAllow": [
        "jira_get_issue",
        "jira_search_issues",
        "jira_list_projects",
        "jira_get_sprint"
      ],
      "disabled": false
    }
  }
}
```

SharePoint MCP — Document & Knowledge Access

Why SharePoint MCP?

Why would an AI assistant need access to SharePoint?

Imagine a project team preparing for an SAP implementation. Important information is spread across SharePoint—project documents, solution designs, statements of work (SOWs), runbooks, meeting documents, and technical guides. A team member asks IBM Bob:

"Find the latest SAP implementation runbook and summarize the key activities required for the upcoming cutover."

Without direct access to SharePoint, Bob would need the user to locate and provide the relevant documents manually. With **SharePoint MCP**, Bob can interact with SharePoint through MCP tools, search for relevant documents, retrieve the required content, and use it to answer the request.

The SharePoint MCP server can expose capabilities such as searching documents, retrieving files, locating project artifacts, and accessing relevant content based on the user's request and permissions.

The interaction can be summarized as:

```
User Request
     ↓
   IBM Bob
     ↓
SharePoint MCP
     ↓
 SharePoint
     ↓
Documents / Project Information
     ↓
   IBM Bob
     ↓
Analysis & Summary
     ↓
User Response
```

Key Takeaways

With SharePoint MCP, an AI assistant can securely access relevant project knowledge and documents, so users can find, retrieve, and understand information through natural-language requests instead of manually searching through document repositories.

Overview

SharePoint is the backbone of document management on most IBM SAP delivery projects — housing SOWs, runbooks, test scripts, architecture diagrams, and project plans. With IBM Bob connected to SharePoint via MCP, you can search, retrieve, and reason over project documents without manual copy-paste.

The following list describes possible SAP delivery use cases:

- Search for the latest SOW or contract amendment
- Retrieve runbooks during incident resolution
- Access test scripts while writing automation
- Pull architecture decisions from project wikis

Is a SharePoint MCP Server Available?

Check first:

1. Settings (⚙️) → MCP tab → search **"SharePoint"** or **"Microsoft"** in the marketplace
2. Check **https://mcpmarket.com** — filter by "Collaboration Tools" category (largest registry, 48,000+ servers)
3. Check https://smithery.ai and https://mcp.so for community servers
4. Check https://github.com/modelcontextprotocol/servers

Likely outcome: As of 2026, a fully certified SharePoint MCP server may not be available in the Bob marketplace. Community options may exist but vary in quality and coverage.

If no suitable server is found → Build a custom MCP server.

Configuration Levels and Opening the Config File

Where Does the Server Config Live?

Before adding the server, decide which level is right for your situation:

Level	File Location	When to Use
Global	~/.bob/mcp.json	Personal tools you want available on every project on your machine
Project	.bob/mcp.json in your project root	Team integrations tied to this engagement — commit to version control so every team member gets the same servers automatically

> **Project-level takes precedence** over global when the same server name exists in both files.

How to Open the Config File in IBM Bob (Step by Step)

1. Open VS Code with your project loaded
2. Click the **IBM Bob** icon in the Activity Bar to open the Bob panel
3. Click the **Settings** icon (⚙️) at the top of the Bob panel
4. Select the **MCP** tab
5. You will see two buttons near the top of the tab:
   - **Edit Global MCP**: opens `~/.bob/mcp.json` *(applies to all your workspaces)*
   - **Edit Project MCP**: opens `.bob/mcp.json` inside your current project folder *(Bob creates the file automatically if it does not exist yet)*
6. Click the appropriate button — the JSON file opens in VS Code
7. Add your server entry under the `mcpServers` object and **save the file**
8. Bob detects the change and **hot-reloads automatically** — no restart needed
9. Switch back to the **MCP** tab — your new server appears in the list within seconds

Setting Environment Variables for Credentials

All credentials must be stored as environment variables — never hardcoded in the JSON file. Reference them as `"${env:VAR_NAME}"` in `mcp.json`.

Windows (PowerShell — persistent):

```
[System.Environment]::SetEnvironmentVariable('MY_API_KEY', 'your-value-here', 'User')
```

macOS / Linux (persistent — add to shell profile):

```
echo 'export MY_API_KEY="your-value-here"' >> ~/.zshrc && source ~/.zshrc
```

> **Important:** After setting a new environment variable, restart VS Code so Bob picks up the new value.
>
> **SharePoint → Use Project** (`.bob/mcp.json`)
>
> SharePoint credentials and site URLs are specific to this delivery engagement. Commit `.bob/mcp.json` to version control so the whole team shares the same configuration. Credentials (`MS_CLIENT_ID`, `MS_CLIENT_SECRET`, `MS_TENANT_ID`) must always be environment variables — never hardcoded in the file.

Option A (Recommended)

Build a Custom SharePoint MCP Server

When no suitable server exists, **build one** using the **Microsoft Graph API** — the official REST API that powers SharePoint, OneDrive, Teams, and all Microsoft 365 services.

This is the recommended approach for enterprise IBM SAP delivery projects because:

- Full control over which sites, libraries, and document types are exposed
- Can be scoped to project-specific SharePoint sites
- Credentials managed centrally via Azure App Registration
- Can be deployed as a shared team MCP server over HTTP transport

Microsoft Graph API — What It Provides

Graph API Endpoint	What It Does
GET /sites/{site-id}/drive/root/children	List files in a document library
GET /sites/{site-id}/drive/items/{item-id}	Get file metadata
GET /drive/items/{item-id}/content	Download file content
GET /search/query	Full-text search across SharePoint
GET /sites/{site-id}/lists	List SharePoint lists
GET /sites/{site-id}/lists/{list-id}/items	Get list items

Azure App Registration (One-Time Setup)

1. Go to **Azure Portal** → **Azure Active Directory** → **App Registrations**
2. Click **New registration**
   - Name: `IBM-Bob-MCP-SharePoint`
   - Supported account types: *Single tenant*
3. After creation, note the **Application (client) ID** and **Directory (tenant) ID**
4. Go to **Certificates & secrets** → **New client secret** — copy the value immediately
5. Go to **API permissions** → **Add a permission** → **Microsoft Graph** → **Application permissions**
   - Add: `Sites.Read.All` (for read-only access)
   - Add: `Files.Read.All`
6. Click **Grant admin consent**

Set these as environment variables:

```
# Windows
[System.Environment]::SetEnvironmentVariable('MS_CLIENT_ID','your-client-id','User')
[System.Environment]::SetEnvironmentVariable('MS_CLIENT_SECRET','your-secret','User')
[System.Environment]::SetEnvironmentVariable('MS_TENANT_ID','your-tenant-id','User')
```

Scaffold the Custom MCP Server

Ask IBM Bob directly:

```
"Build an MCP server that connects to Microsoft SharePoint using the Graph API.
 It should expose tools to:
 1. Search documents across a SharePoint site by keyword
 2. List files in a document library
 3. Get the text content of a specific document
 Use environment variables MS_CLIENT_ID, MS_CLIENT_SECRET, MS_TENANT_ID, and SHAREPOINT_SITE_URL."
```

Bob will:

- Scaffold a TypeScript/Node.js MCP project
- Implement the tools using `@modelcontextprotocol/server` and `@azure/msal-node`
- Add the server to your `mcp.json` with environment variable references
- Connect automatically on completion

Register in mcp.json

Once Bob builds and compiles the server:

```
{
  "mcpServers": {
    "sharepoint": {
      "command": "node",
      "args": ["/path/to/sharepoint-mcp/build/index.js"],
      "env": {
        "MS_CLIENT_ID": "${env:MS_CLIENT_ID}",
        "MS_CLIENT_SECRET": "${env:MS_CLIENT_SECRET}",
        "MS_TENANT_ID": "${env:MS_TENANT_ID}",
        "SHAREPOINT_SITE_URL": "https://yourcompany.sharepoint.com/sites/SAP-Project"
      },
      "alwaysAllow": [
        "search_documents",
        "list_files",
        "get_document_content"
      ]
    }
  }
}
```

Option B (Community MCP)

Use a Community SharePoint / Microsoft 365 MCP Server

If a community package is available (search npm for `mcp-server-sharepoint` or `mcp-microsoft365`):

```
{
  "mcpServers": {
    "sharepoint": {
      "command": "npx",
      "args": ["-y", "mcp-server-sharepoint"],
      "env": {
        "SHAREPOINT_SITE_URL": "https://yourcompany.sharepoint.com/sites/SAP-Project",
        "MS_CLIENT_ID": "${env:MS_CLIENT_ID}",
        "MS_CLIENT_SECRET": "${env:MS_CLIENT_SECRET}",
        "MS_TENANT_ID": "${env:MS_TENANT_ID}"
      }
    }
  }
}
```

Key Tools to Implement

When building or configuring your SharePoint MCP server, ensure it exposes these tools:

Tool	Description
search_documents	Full-text search across the SharePoint site
list_files	List files in a specific document library or folder
get_document_content	Retrieve plain text content of a document
get_file_metadata	Get file name, size, author, last modified
list_sites	List available SharePoint sites
get_list_items	Retrieve items from a SharePoint list

Example Prompts for SAP Projects

```
1. Search SharePoint for the latest SOW for the SAP S/4HANA migration project

2. List all documents in the 'Test Scripts' library of our project site

3. Get the content of the document named 'Cutover Runbook v3.docx'

4. Find any architecture decisions related to the finance module

5. What does our runbook say about rolling back a failed transport?
```

Troubleshooting

Problem	Cause	Fix
401 Unauthorized	Wrong client ID / secret	Re-check Azure App Registration values
403 Forbidden	Admin consent not granted	Grant admin consent in Azure Portal
Site not found	Wrong site URL	Use /sites endpoint to list available sites
Empty search results	Site not indexed	Allow 24h for new SharePoint content to index
Large documents truncated	Token limit	Request specific sections or page the results

Summary

Scenario	Approach
Community server found in marketplace	Install via Bob marketplace
Community npm package available	Configure manually in mcp.json
No server available	Build custom using Microsoft Graph API (see Building a Custom MCP Server)

Building a custom SharePoint MCP server is straightforward, gives full control, and ensures enterprise-grade security through Azure App Registration and least-privilege permissions.

IBM Bob and GitHub: AI-Assisted Version Control and CI/CD for SAP Delivery

IBM Bob and GitHub: AI-Assisted Version Control and CI/CD for SAP Delivery

> **Disclaimer:** All examples in this section are for demonstration purposes only. The projects, change request numbers, repository names, and pipeline configurations shown here are illustrative samples. They do not represent standard projects, templates, or recommended implementations supplied by IBM, SAP, or any other vendor.

GitHub is not just a place where code lives. In an AI-assisted SAP delivery program, it is the connective tissue that links requirements, development, review, and deployment into a single auditable chain. When IBM Bob participates in that chain, the quality of what goes into GitHub improves, and the effort required to get it there drops substantially.

This section covers two distinct but complementary ways that Bob works with GitHub. The first is through the GitHub MCP Server — an official integration that gives Bob programmatic access to every major GitHub API, from creating repositories to managing pull requests and running searches across codebases. The second is through Bob's built-in Git tooling, which lets developers stage, commit, branch, push, and raise pull requests entirely through natural language, without leaving the IDE.

Both patterns apply directly to SAP development. The section walks through complete worked examples for a SAP UI5 application and a SAP ABAP codebase managed with abapGit, and shows how the resulting pull requests feed into GitHub Actions CI/CD pipelines that run linting, building, and compliance checks automatically.

GitHub in the SAP Software Delivery Lifecycle

Most SAP delivery teams treat Git as a deployment mechanism — something that holds transport-equivalent artifacts and gets synced to the target system via abapGit or SAP Business Application Studio. That is a reasonable starting point, but it captures only a fraction of what GitHub can do when it is properly connected to the development toolchain.

GitHub becomes genuinely useful in SAP delivery when it serves all four of the following functions simultaneously: a version-controlled repository for code and configuration, a collaboration platform for code review and pull requests, an automation host for CI/CD pipelines, and an audit trail that links commits and pull requests back to ServiceNow change requests. IBM Bob is the bridge that makes all four functions work together without requiring developers to context-switch between tools.

What GitHub Brings to SAP Projects

For SAP UI5 development, GitHub provides the same infrastructure it provides for any web project: linting pipelines, build automation, branch protection rules, and code review workflows. For SAP ABAP development, GitHub — combined with abapGit — provides something that was historically difficult to achieve in SAP landscapes: proper version control with a human-readable diff for every program change.

The abapGit serialization format stores each ABAP program as a pair of files: a `.prog.abap` file containing the source code and a `.prog.xml` file containing the metadata. Both files are plain text, which means GitHub's diff, blame, and history tools work on them exactly as they would on any other source file. A reviewer can see precisely which lines changed, why they changed, and which transport and change request they belong to.

The Value IBM Bob Adds

The honest reality of most SAP Git workflows without AI assistance is that the commit discipline is inconsistent. Developers write terse or uninformative commit messages, forget to link commits to change requests, raise pull requests with empty descriptions, and skip linting steps when deadlines are close. IBM Bob addresses all of these failure modes directly — not by enforcing process through tooling restrictions, but by making the right thing the easy thing.

When Bob generates commit messages from the actual diff, they are consistent, informative, and correctly formatted. When Bob raises pull requests, the descriptions include the test steps, the ServiceNow traceability fields, and the compliance checklist that reviewers actually need. None of this requires the developer to remember a template or consult a style guide.

Connecting IBM Bob to GitHub

There are two integration pathways between IBM Bob and GitHub. They operate at different levels of the stack and serve different purposes, but they are complementary and can both be active simultaneously.

Authentication: Personal Access Tokens

The primary authentication mechanism for programmatic GitHub access is the Personal Access Token (PAT). A PAT is a credential generated in GitHub's developer settings that grants a defined scope of access to one or more repositories. For IBM Bob, the PAT is stored as a local environment variable — it never appears in source code or is transmitted outside the configured scope.

When setting up a PAT for Bob, the following decisions matter:

```
# Recommended PAT scope for IBM Bob GitHub integration
repo          — read/write access to code, commits, branches, pull requests
read:org      — read organization membership (for org-scoped repos)
workflow      — ability to trigger GitHub Actions workflows

# Store in Bob environment configuration:
GITHUB_TOKEN=<your-personal-access-token>

# Never commit this value to source control.
# Use .env files (gitignored) for local development,
# or Code Engine / Kubernetes secrets for deployed environments.
```

The following steps show how to create a fine-grained Personal Access Token in GitHub and register it with IBM Bob.

Step 1: Navigate to GitHub token settings

Navigate to **GitHub Settings** → **Developer Settings** → **Personal Access Tokens** → **Fine-grained tokens** → **Generate new token**. Choose the target repositories, set a 90-day expiry, and grant the permissions listed above (Repository: Read/Write for Code and Pull Requests; Workflow: Read/Write).

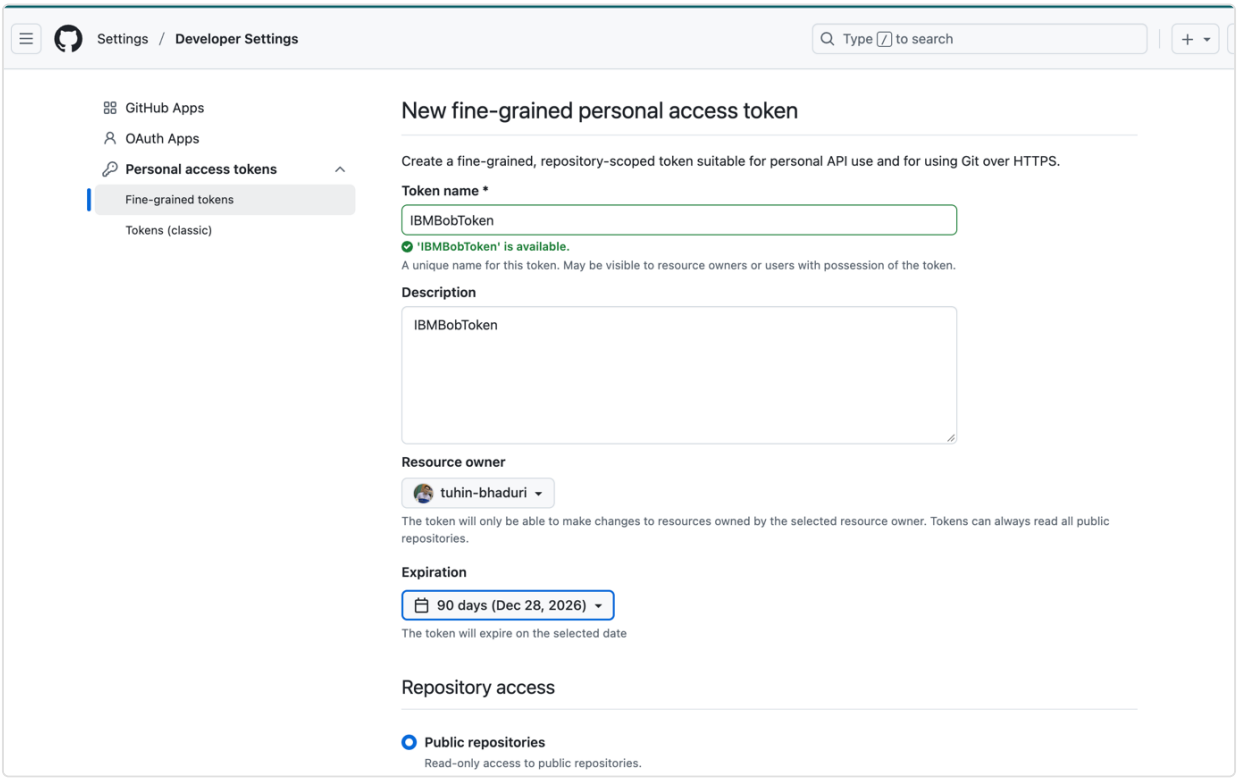


Figure: GitHub fine-grained personal access token creation form

Once you click Generate Token, GitHub shows the token value exactly once. Copy it immediately — you will not be able to view it again.

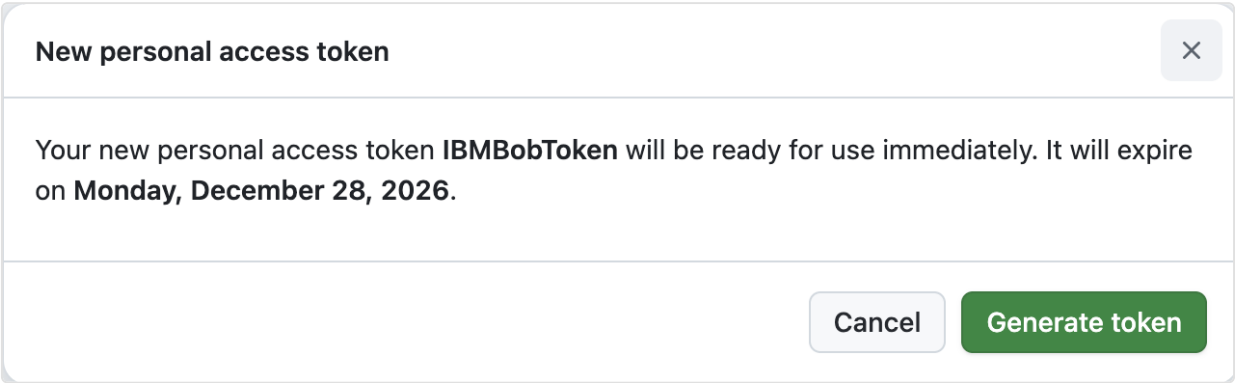


Figure: GitHub personal access token confirmation screen

Step 2: Store the token as an environment variable

Store the token as a shell environment variable so IBM Bob can pick it up without the value ever appearing in source code. Add the following export to your shell profile:

```
# Add to ~/.zshrc or ~/.bashrc
echo 'export GITHUB_TOKEN="your-new-github-pat-here"' >> ~/.zshrc && source ~/.zshrc
```

Step 3: Reference the variable in mcp.json

Reference the environment variable in the IBM Bob MCP configuration file (`.bob/mcp.json`) using the `${env:VARIABLE_NAME}` syntax. This means the PAT is resolved from the environment at runtime and never stored in the config file itself:

```
"github": {
  "command": "npx",
  "args": ["-y", "@modelcontextprotocol/server-github"],
  "env": {
    "GITHUB_PERSONAL_ACCESS_TOKEN": "${env:GITHUB_TOKEN}"
  }
}
```

> **Important:** Treat a GitHub PAT as you would a password. Set the minimum required scope, use a short expiry (90 days is a reasonable default), and rotate it when the expiry date approaches. A PAT with the full `repo` scope on a production organization is a high-value credential.

The GitHub MCP Server

Beyond the built-in Git tooling, Bob can connect to GitHub through the official GitHub MCP Server — a Model Context Protocol integration that exposes the full GitHub API as a set of callable tools. The GitHub MCP Server is available from the IBM Bob MCP Marketplace and can be installed directly from within Bob.

Once installed, the GitHub MCP Server gives Bob the ability to create and search repositories, read and write file contents, manage branches and pull requests, work with issues, and search across code — all through natural language, without any direct API calls from the developer. The server handles authentication, request formatting, and error handling transparently.

> **Tip:** The GitHub MCP Server and Bob's built-in Git tooling serve different purposes. The built-in tooling is optimized for the commit-push-PR workflow on the local repository you are actively working in. The MCP Server is better suited for cross-repository operations, search, issue management, and automation tasks that span multiple repos.

GitHub MCP Capability Map

The following tables describe the tools available through the GitHub MCP Server, organized by functional area.

Repository Management

Tool / API	What it does
create_repository	Create a new GitHub repository with specified settings
search_repositories	Search across GitHub repositories by keyword, topic, or filter
fork_repository	Fork an existing repository to your account or organization

File Operations

Tool / API	What it does
get_file_contents	Read the contents of a file or list a directory
create_or_update_file	Create a new file or update an existing one, with an auto-generated commit
push_files	Push multiple files to a branch in a single commit

Branches and Commits

Tool / API	What it does
create_branch	Create a new branch from a specified base
list_commits	List and filter commits on any branch
update_pull_request_branch	Sync a PR branch with the latest changes from its base branch

Pull Requests

Tool / API	What it does
create_pull_request	Create a new pull request with title, body, base, and head
list_pull_requests	List and filter pull requests by state, author, label, or base branch
get_pull_request	Get the full detail of a specific pull request
get_pull_request_files	List all files changed in a pull request
get_pull_request_status	Get the combined check status for a pull request
get_pull_request_comments	Retrieve all review comments on a pull request
get_pull_request_reviews	Retrieve all reviews (approvals, rejections, comments)
create_pull_request_review	Create a review: approve, request changes, or comment
merge_pull_request	Merge a pull request using merge, squash, or rebase strategy

Issues

Tool / API	What it does
create_issue	Create a new issue with title, body, labels, and assignees
list_issues	List issues with filtering by state, label, assignee, or milestone
get_issue	Get the full detail of a specific issue
update_issue	Update an existing issue's title, body, state, or labels
add_issue_comment	Add a comment to an issue

Search

Tool / API	What it does
search_code	Search for code across all accessible repositories
search_issues	Search for issues and pull requests by content or metadata
search_users	Search for users by name, email, or organization membership

IBM Bob's Built-In Git Workflow

IBM Bob integrates with VS Code's Source Control panel and adds a natural-language layer on top of standard Git commands. The result is a three-tier architecture in which the developer expresses intent in plain English, Bob translates that intent into Git operations, and Git executes those operations against the local repository and remote.

The Source Control icon in the VS Code Activity Bar shows a badge with the count of files that have pending changes. Clicking it opens the Source Control panel where IBM Bob surfaces all Git state and actions.

The Three-Layer Architecture

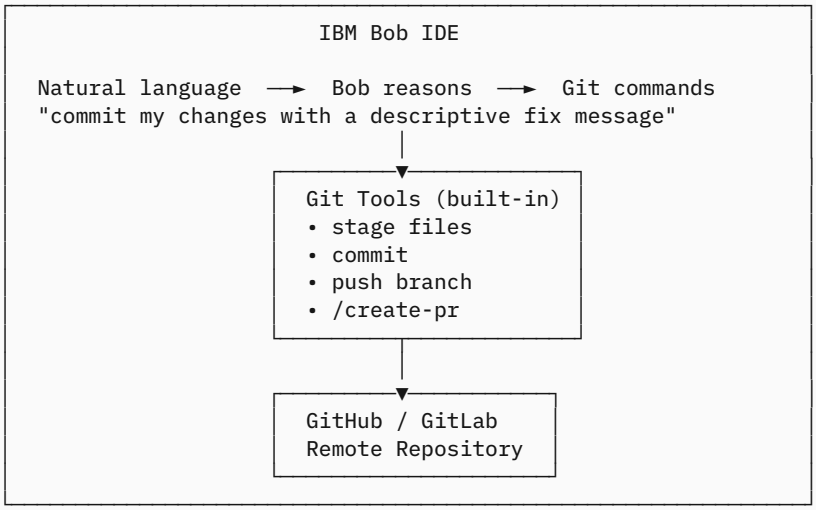




Figure: Git Graph VS Code extension icon

The Source Control Panel at a Glance

IBM Bob surfaces Git state directly in the VS Code Source Control panel, making it easy to see what has changed and take action without opening a terminal. The panel uses three status indicators to classify each changed file:

Table: Source Control panel status indicators

Status	Meaning
M (Modified)	The file existed before and has been changed — ready to stage and commit
U (Untracked)	A newly created file not yet tracked by Git — must be staged before it can be committed
D (Deleted)	The file has been removed from the working tree — the deletion must be staged to record it in Git history

The following screenshot shows the Source Control CHANGES panel after a typical IBM Bob-assisted development session. Modified files are grouped at the top, followed by untracked additions. The panel provides a one-click path to staging individual files or all changes at once.

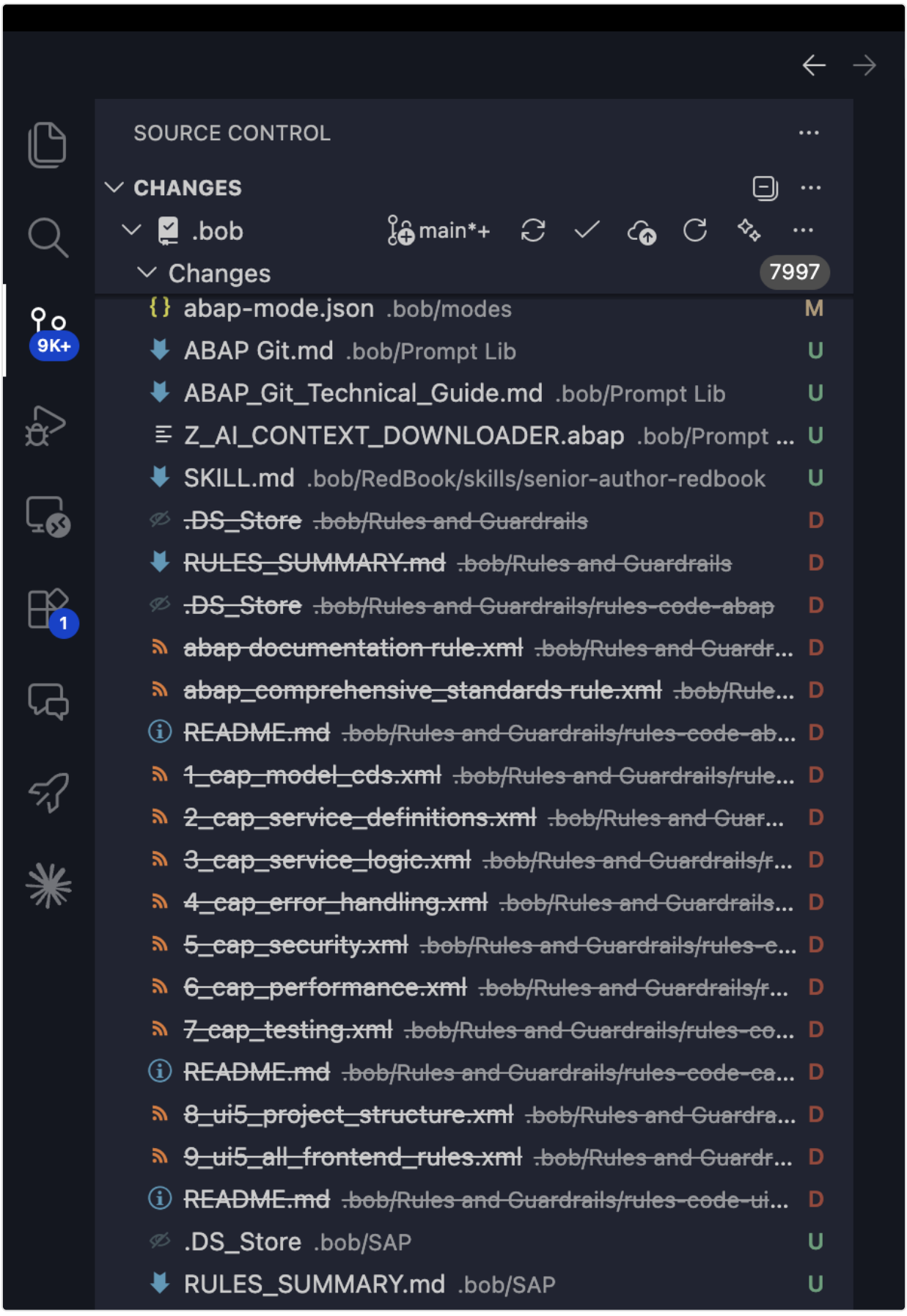

SOURCE CONTROL
CHANGES
.bob
main*+
Changes
7997
abap-mode.json .bob/modes M
ABAP Git.md .bob/Prompt Lib U
ABAP_Git_Technical_Guide.md .bob/Prompt Lib U
Z_AI_CONTEXT_DOWNLOADER.abap .bob/Prompt ... U
SKILL.md .bob/RedBook/skills/senior-author-redbook U
.DS_Store .bob/Rules and Guardrails D
RULES_SUMMARY.md .bob/Rules and Guardrails D
.DS_Store .bob/Rules and Guardrails/rules-code-abap D
abap-documentation-rule.xml .bob/Rules and Guardr... D
abap_comprehensive_standards-rule.xml .bob/Rule... D
README.md .bob/Rules and Guardrails/rules-code-ab... D
1_cap_model_cds.xml .bob/Rules and Guardrails/rule... D
2_cap_service_definitions.xml .bob/Rules and Guar... D
3_cap_service_logic.xml .bob/Rules and Guardrails/r... D
4_cap_error_handling.xml .bob/Rules and Guardrails... D
5_cap_security.xml .bob/Rules and Guardrails/rules-c... D
6_cap_performance.xml .bob/Rules and Guardrails/r... D
7_cap_testing.xml .bob/Rules and Guardrails/rules-co... D
README.md .bob/Rules and Guardrails/rules-code-ca... D
8_ui5_project_structure.xml .bob/Rules and Guardra... D
9_ui5_all_frontend_rules.xml .bob/Rules and Guardr... D
README.md .bob/Rules and Guardrails/rules-code-ui... D
.DS_Store .bob/SAP U
RULES_SUMMARY.md .bob/SAP U

Figure: VS Code Source Control panel with changed ABAP files

Once files are staged, the panel shows them in a separate Staged Changes section. The Git operations dropdown (the ellipsis menu in the panel header) provides access to branch management, stash, pull, push, and other Git commands — all without leaving the IDE.

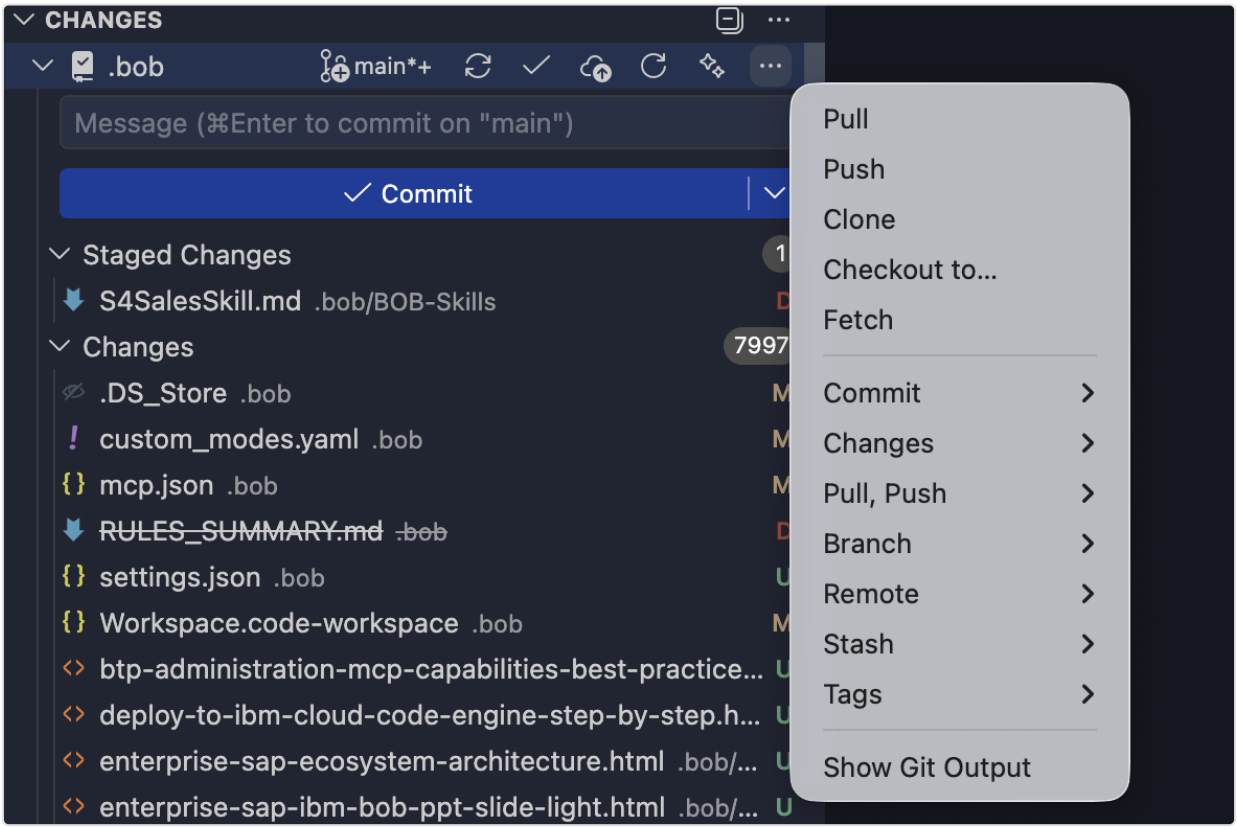


Figure: VS Code staged changes with the Git operations menu open

IBM Bob also integrates with the commit message editor (`COMMIT_EDITMSG`). When the developer asks Bob to suggest a commit message, Bob generates it and pre-populates the editor. The editor shows the staged files and the diff of changes not yet staged alongside the commit message — giving the developer full context when reviewing or editing the generated message before confirming.

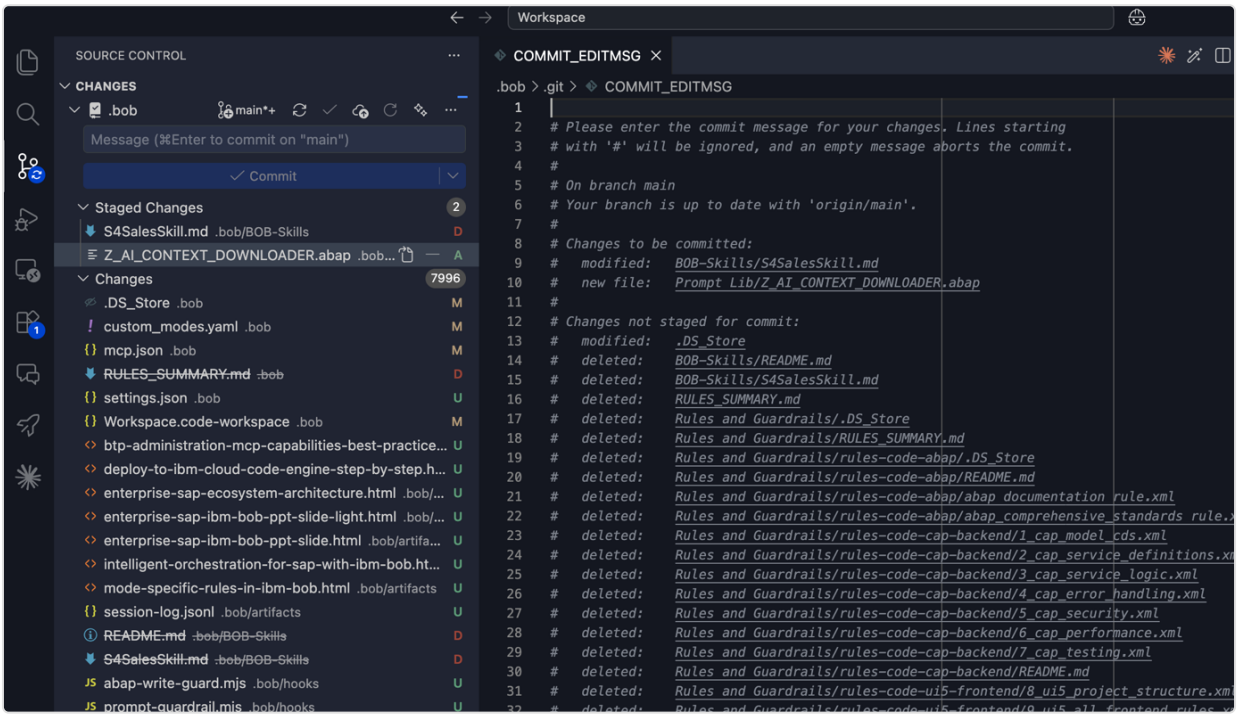


Figure: VS Code commit message editor pre-populated by IBM Bob

The Git Graph View

IBM Bob includes a GRAPH view that renders the full commit history of the repository as a visual branch graph. The graph shows each commit as a node on its branch, with lines connecting merges and branch points. This makes it straightforward to understand the sequence of changes, identify where branches diverged, and trace specific commits back to their pull requests and change requests.

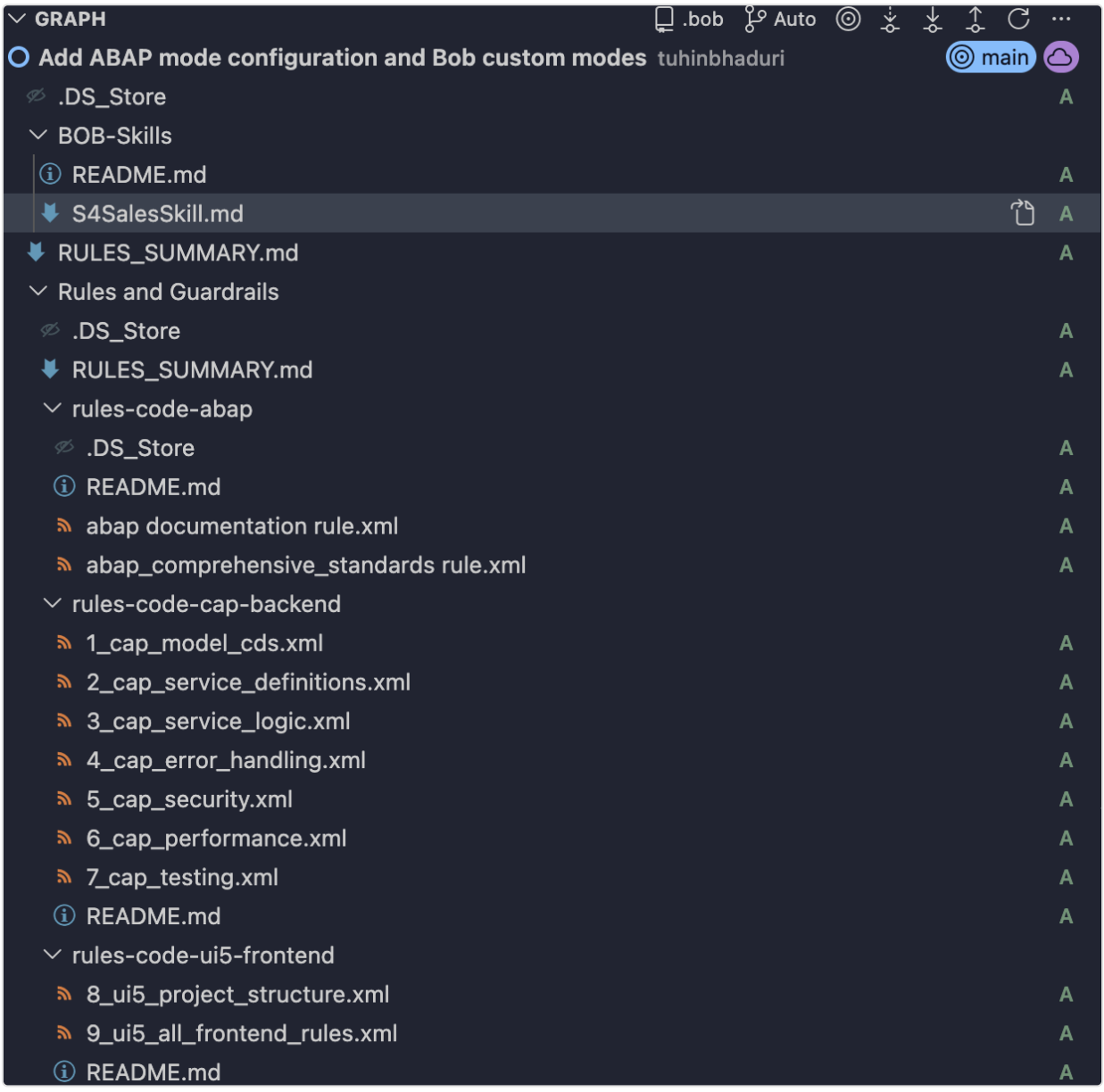


Figure: VS Code Git Graph commit history tree

The GRAPH view is particularly useful during code reviews, when a reviewer wants to understand the commit sequence behind a pull request without switching to GitHub. Selecting any commit node shows the files changed and the diff inline — the same information available in GitHub, but accessible without leaving the IDE.

Git Context Mentions and Slash Commands

Bob supports two special references that load live Git state into the conversation context, plus a slash command that triggers the full PR creation flow.

Table: Bob Git context mentions and slash commands

Reference / Command	What Bob can do with it
@git-changes	Loads the full git status and diff of all uncommitted changes. Use this before committing or reviewing.
@	Loads the commit message, author, date, and full diff for a specific past commit.
/create-pr	Triggers the interactive AI-assisted pull request creation flow (base branch selection → title + description generation → review → PR submission).

Natural Language Trigger Reference

Table: Natural language triggers and their Git equivalents

You say…	Bob does…	Git command(s) executed
"Stage all my changes"	Stages all modified and new files	git add .
"Stage changes in webapp/"	Stages only the specified directory	git add webapp/
"Create a branch called feature/X"	Creates and checks out the new branch	git checkout -b feature/X
"Suggest a commit message for @git-changes"	Generates a conventional commit message	(no command yet — awaiting approval)
"Commit with that message"	Commits the staged changes	git commit -m "<generated>"
"Push my branch to origin"	Pushes and sets upstream tracking	git push -u origin <branch>
"Review @git-changes for Clean ABAP"	Reads diff, reports policy violations	(analysis only — no git command)
"Fix the ESLint errors from the CI output"	Reads CI log, applies fixes, re-lints	(code edits + npm run lint)

Walkthrough: SAP UI5 Project

> **Note:** The project used in this walkthrough — SalesOrderDetailApp — is a demonstration application built on OpenUI5 1.120. It is not a production application and does not represent a standard IBM or SAP deliverable. It is used here to illustrate the Git workflow pattern.

This section walks through the complete IBM Bob Git workflow for a SAP UI5 project, from repository initialization to a merged pull request. The same pattern applies to any SAP UI5 or SAP Fiori application.

Project Overview

```
sales.order.detail.app  (OpenUI5 1.120.0)

├── webapp/
│   ├── Component.js
│   ├── manifest.json
│   ├── controller/
│   ├── view/            ← XML views: App, Main, Detail
│   ├── model/
│   └── i18n/
├── ui5.yaml             ← specVersion 3.0, OpenUI5 1.120.0
└── package.json         ← scripts: start | build | lint | test
```

Step-by-Step Workflow

Step 1: Initialize repository and make first commit

```
You → Bob: **Initialize a Git repository for the project, create a .gitignore that excludes node_modules a
```

Bob creates the `.gitignore`, stages all files, and makes the initial commit with a correctly formatted conventional commit message. The developer does not need to open a terminal.

Step 2: Create a feature branch

```
You → Bob: **Create a new branch called feature/sales-order-detail-enhancements**
```

Bob runs `git checkout -b feature/sales-order-detail-enhancements`. The developer is now on an isolated branch and can make changes without affecting main.

Step 3: Understand and stage changes

```
 You → Bob: **Review @git-changes and summarize what I changed**
```

Bob reads the full diff from `@git-changes` and returns a structured change summary — which files changed, and what each change does at the business logic level. After review, the developer asks Bob to stage the changes and Bob runs the appropriate `git add` commands.

Step 4: Generate a conventional commit message

```
You → Bob: **Suggest a commit message for @git-changes**
```

Bob analyzes the diff and generates a commit message following the conventional commits standard (`type(scope): description`). An example output for the demo project:

```
fix(detail): add delivery status display and fix OData expand for items

- Added ObjectStatus control to Detail.view.xml for status binding

- Fixed expand parameter in Detail.controller.js for line item loading

- Follows SAP UI5 Fiori Object Page pattern
```

Step 5: Validate code quality before pushing

```
You → Bob: **Run the lint check on the UI5 app**
```

Bob runs `npm run lint` (which invokes ESLint against the `webapp/` directory). Any violations are reported inline in the chat. If issues are found, the developer can immediately ask Bob to fix them and Bob applies the corrections, then re-runs the linter to confirm a clean result. No terminal switching is required.

Step 6: Push the branch to the remote

```
You → Bob: **Push my branch to origin**
```

Step 7: Create the pull request

```
You → Bob: **/create-pr**
```

The `/create-pr` command triggers Bob's interactive PR creation flow. Bob asks the developer to select the base branch, then analyzes the branch name, commit history, and full diff to generate a PR title and description. The generated description for the demo project includes:

```
## Summary

Enhances the Detail page with delivery status visibility and fixes the

OData expand parameter for order item drill-down.

## Changes

- webapp/view/Detail.view.xml: Added ObjectStatus control for status display

- webapp/controller/Detail.controller.js: Corrected $expand parameter

## Test Steps

1. npm start → navigate to any Sales Order

2. Confirm delivery status badge appears on the Object Page header

3. Click Detail section → confirm items load without additional fetch
```

The developer can review the generated description inline, ask Bob to refine it, and approve it when satisfied. Bob then creates the PR and returns the direct link in the chat.

The following screenshot shows the Create Pull Request dialog with Bob's four-step completion checklist — all steps confirmed green before the PR is submitted. This ensures that base branch selection, description generation, review, and final confirmation all complete successfully before the PR is created.



Figure: IBM Bob Create Pull Request dialog with all steps completed

> **Tip:** The `/create-pr` flow is iterative. If the generated description is missing context, simply type your feedback in chat ("Add a note about the Fiori Design Guidelines" or "The test steps need a pre-condition about the user role") and Bob regenerates the description immediately. There is no limit to the number of iterations.

Walkthrough: SAP ABAP Project (abapGit)

> **Note:** The ABAP programs used in this walkthrough are demonstration programs managed with abapGit. They are not production-grade programs and do not represent standard IBM or SAP deliverables. The change request numbers, transport IDs, and program names are illustrative only.

For SAP ABAP development, the Git workflow follows the same Bob pattern but with ABAP-specific review rules and a branch naming convention that ties each branch to a ServiceNow change request. This creates a direct link between the Git history and the enterprise change management record.

The abapGit File Structure

abapGit serializes ABAP programs into a pair of plain-text files that Git can version control like any other source code:

```
SAP ABAP/ABAP/src/

├── package.devc.xml              ← ABAP package definition
├── z<programname>.prog.abap       ← ABAP source code
├── z<programname>.prog.xml        ← Program metadata and properties
└── ...                           ← Additional programs follow the same pattern
```

Because both files are plain text, GitHub's diff tools work on them exactly as they do on JavaScript or Python. A reviewer can see which lines changed, read the diff in context, and approve or reject the change without needing access to the SAP system.

Step-by-Step Workflow

Step 1: Initialize the abapGit repository

```
You → Bob: **Initialize a Git repository for the ABAP project, add a .gitignore for abapGit projects, and
```

Step 2: Create a change-request-linked branch

```
You → Bob: **Create a branch called feature/CHG0000003-top5-sales-orders**
```

Embedding the change request number in the branch name creates a direct trace from the Git branch to the ServiceNow record. Automated CI/CD checks can then verify that every commit on the branch references the correct `CHG` number. This pattern makes the audit trail explicit without requiring a separate tracking document.

Step 3: Review ABAP changes for compliance

```
You → Bob: **Review @git-changes in the ABAP source and check for any Clean ABAP violations**
```

Bob reads the diff and checks the ABAP source against a set of best-practice rules drawn from the Clean ABAP guidelines and S/4HANA development standards. An example review output:

```
✅ SELECT uses explicit field list — compliant

✅ FOR ALL ENTRIES guarded by IS NOT INITIAL check — compliant

✅ AUTHORITY-CHECK present for the relevant authorization object — compliant

⚠️  Line 352: String template uses obsolete concatenation style

    → Consider |{ variable }| syntax per Clean ABAP guidelines
```

The developer can immediately ask Bob to fix the flagged issue. Bob applies the correction and shows the updated snippet for confirmation.

Step 4: Generate a change-request-linked commit message

```
You → Bob: **Suggest a commit message for @git-changes referencing the change request number**
```

Bob generates a commit message that embeds the change request number, describes the change in plain English, and includes the compliance status. The traceability field ensures CI/CD checks can verify the link:

```
feat(ABAP): CHG0000003 — add customer name column to sales order report

- Joined customer table to order header SELECT for name retrieval

- Added customer name column to internal type and ALV configuration

- Applied string template fix per Clean ABAP guidelines

- Authorization check retained for relevant authorization object

Transport: <transport-number>
```

Step 5: Push and raise the pull request

```
You → Bob: **Push this branch and create a PR**
```

Bob pushes the branch and the `/create-pr` flow generates a PR description that includes the ServiceNow traceability section — change request number, transport reference, assigned developer — along with the abapGit sync test steps and the compliance checklist. The generated description is structured to give the reviewer everything they need for an informed approval decision.

CI/CD Pipeline Integration

Once IBM Bob raises a pull request, the CI/CD pipeline takes over automatically. The pipeline validates the change before it can be merged, enforcing the code quality and traceability standards that a human reviewer might miss or might not have time to check manually.

The following pipeline patterns are illustrative examples. They show what a CI/CD configuration for SAP UI5 and SAP ABAP might look like when integrated with IBM Bob's pull request workflow. Actual implementations will vary based on the organization's toolchain, SAP system landscape, and governance requirements.

SAP UI5 CI/CD Pipeline

For UI5 projects, the pipeline runs on every pull request that touches application code. It installs dependencies, runs ESLint, builds the application, and uploads the build artifact. Bob's role in the pipeline is upstream: it raises the PR that triggers the pipeline, and it can read the pipeline output and apply fixes if the lint step fails.

```
*# .github/workflows/ui5-ci.yml*
name: UI5 CI
on:
  pull_request:
    branches: [main, develop]
    paths:
      - "webapp/**"
      - "ui5.yaml"
      - "package.json"
jobs:
  lint-and-build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: "20"
      - name: Install dependencies
        run: npm ci
      - name: Run ESLint
        run: npm run lint
      - name: Build UI5 application
        run: npm run build
      - name: Upload build artifact
        uses: actions/upload-artifact@v4
        with:
          name: ui5-dist
          path: dist/
```

If the ESLint step fails, the developer can paste the CI output into the Bob chat and ask: "Fix the ESLint errors from the CI output." Bob reads the error log, locates the failing lines, applies the fixes, and re-lints locally before the developer pushes the corrected commit to the PR branch.

SAP ABAP CI/CD Pipeline (abapGit)

For ABAP projects, the pipeline focuses on two checks that are difficult to enforce manually at scale: static code analysis with abaplint, and traceability verification that every commit references a valid change request number. Together these two checks enforce the code quality and governance standards that the organization requires.

```
*# .github/workflows/abap-ci.yml*
name: ABAP CI
on:
  pull_request:
    branches: [main]
    paths:
      - "src/**"
jobs:
  abap-lint:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Run abaplint
        uses: abaplint/abaplint-action@v1
        with:
          config: .abaplint.json
  traceability-check:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Verify change request reference in commit message
        run: |
          git log --oneline HEAD^..HEAD | grep -E "CHG[0-9]+" || \\
            (echo "ERROR: Commit must reference a CHG number" && exit 1)
```

Because IBM Bob embeds the CHG number in every commit message it generates, the traceability check passes without any additional developer effort. The discipline is built into Bob's output format, not enforced as an afterthought.

> **Note:** The abaplint configuration file (`.abaplint.json`) defines which Clean ABAP rules are enforced in the pipeline. The rules Bob checks during the review step in the IDE should match the rules in `.abaplint.json`. Maintaining that alignment means that issues Bob catches in the IDE will not resurface as CI failures, and CI failures will prompt Bob reviews that correctly identify the root cause.

End-to-End Deployment Demo: IBM Bob, SAP CI/CD, and SAP BTP Cloud Foundry

> **Attention:** All steps in this section are for demonstration purposes only. The repository names, credentials, Cloud Foundry endpoints, organization names, and application details shown here are illustrative examples. They do not represent standard projects, configurations, or recommended implementations supplied by IBM, SAP, or any other vendor.

The previous sections described how IBM Bob raises pull requests and how GitHub Actions pipelines validate code quality. This section takes that chain one step further and shows how to connect IBM Bob's Git workflow directly to SAP's own Continuous Integration & Delivery service — so that every commit IBM Bob pushes to GitHub automatically triggers a build and deploys the application to SAP BTP Cloud Foundry.

The methodology shown here applies to any GitHub or GitLab repository and to any CAP or ABAP application managed with abapGit. The end state is a fully automated delivery chain: a developer makes a change inside IBM Bob, and the application is running on BTP without any further manual action.

The complete flow is shown in the following figure:

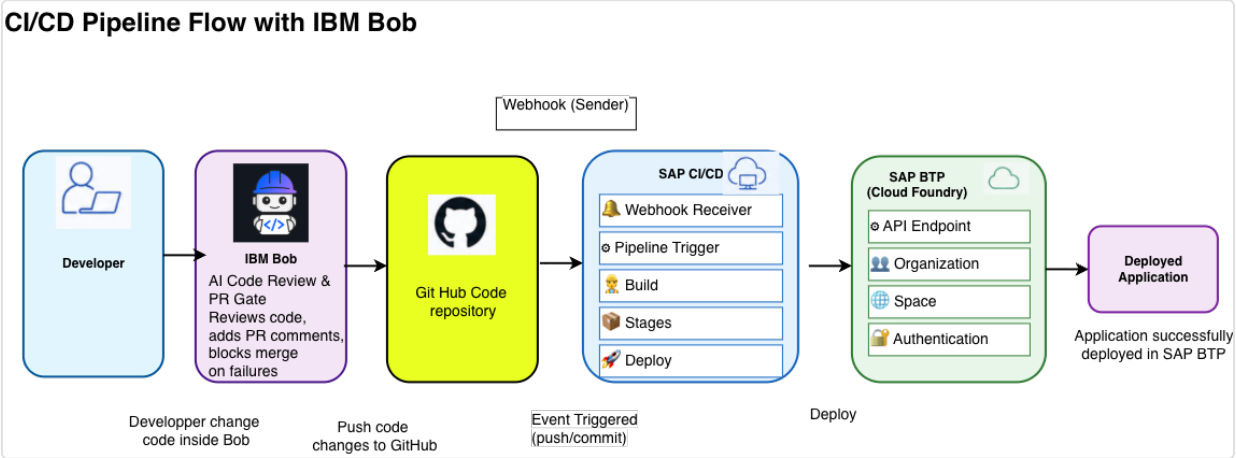


Figure: CI/CD pipeline flow from IBM Bob code change to SAP BTP deployment

There are five stages to set up this pipeline. The following diagram summarizes them at a glance before the step-by-step walkthrough that follows.

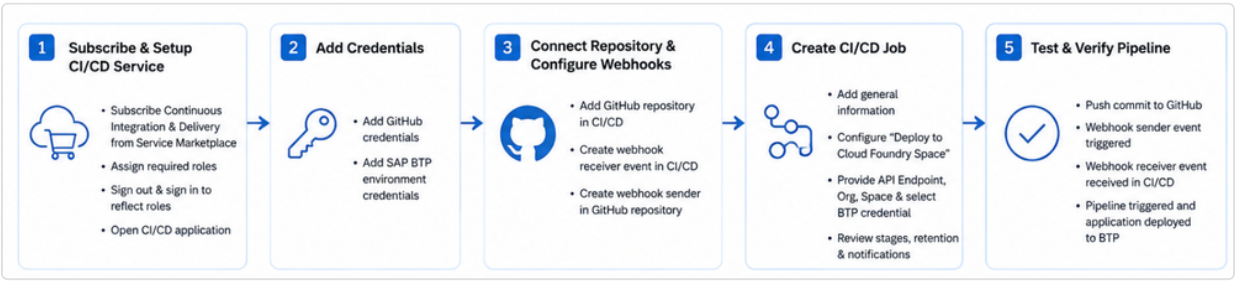


Figure: Five-stage CI/CD setup summary

Step 1: Create a GitHub Access Token for CI/CD

A dedicated Personal Access Token (PAT) should be created specifically for CI/CD operations. Naming it clearly — for example, IBMBobCICD — makes it straightforward to identify, rotate, and revoke independently from the token used for IBM Bob's MCP integration. Follow the same fine-grained token setup described in the **Authentication: Personal Access Tokens** section.

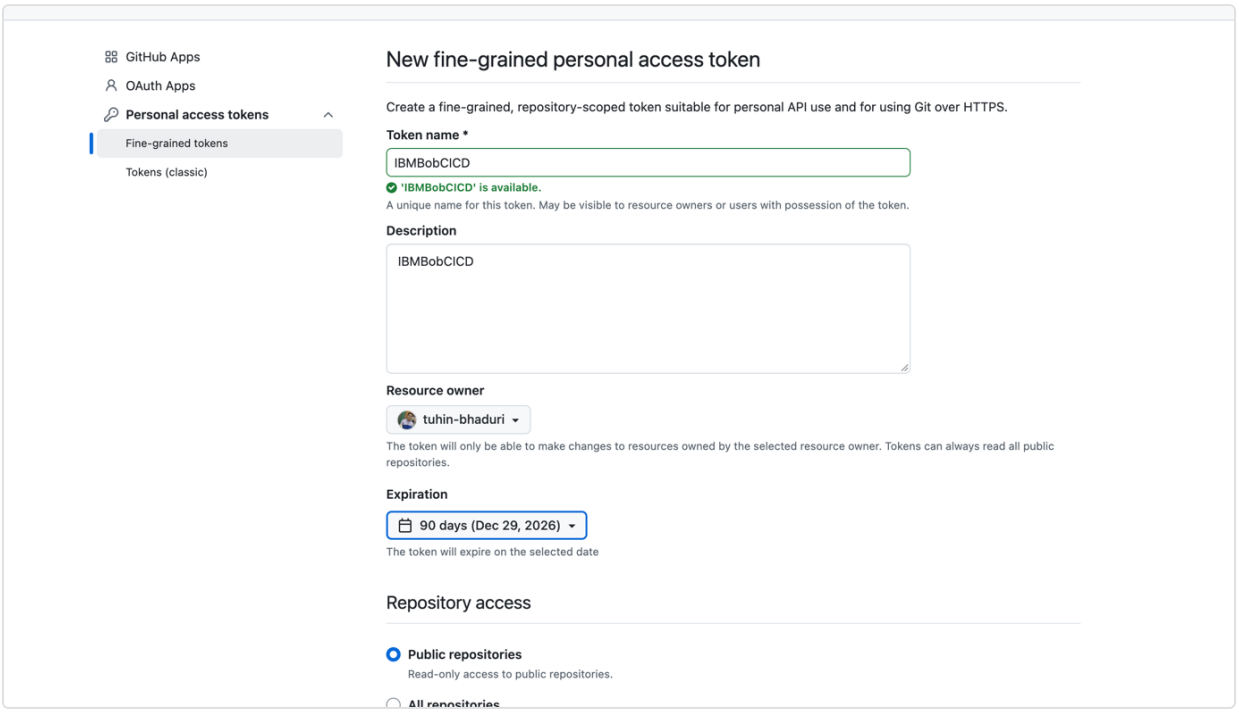


Figure: GitHub PAT creation form for the IBMBobCICD token

Step 2: Configure IBM Bob for IBM GitHub

If you are working with IBM GitHub (github.ibm.com) rather than public GitHub, the MCP Server configuration in `.bob/mcp.json` must include the `GITHUB_HOST` environment variable pointing to the IBM GitHub instance. The `GITHUB_PERSONAL_ACCESS_TOKEN` value is still resolved from the local environment — it is never stored in the configuration file.



Figure: .bob/mcp.json GitHub MCP server configuration for IBM GitHub Enterprise

> **Note:** The `${env:GITHUB_TOKEN}` syntax in `.bob/mcp.json` tells IBM Bob to read the token from the `GITHUB_TOKEN` shell environment variable at runtime. This means the same configuration file can be safely committed to source control — the actual credential value never appears in any file.

Step 3: Subscribe to SAP Continuous Integration & Delivery

SAP Continuous Integration & Delivery is a service in the SAP BTP Service Marketplace. Before it can be used, it must be subscribed to at the subaccount level, and the appropriate role collections must be assigned to the users who will configure and operate the pipelines.

Subscribe to the Service

Navigate to the BTP Cockpit and open the Service Marketplace for your subaccount. Search for Continuous Integration & Delivery and create a new subscription using the trial (or standard) plan.

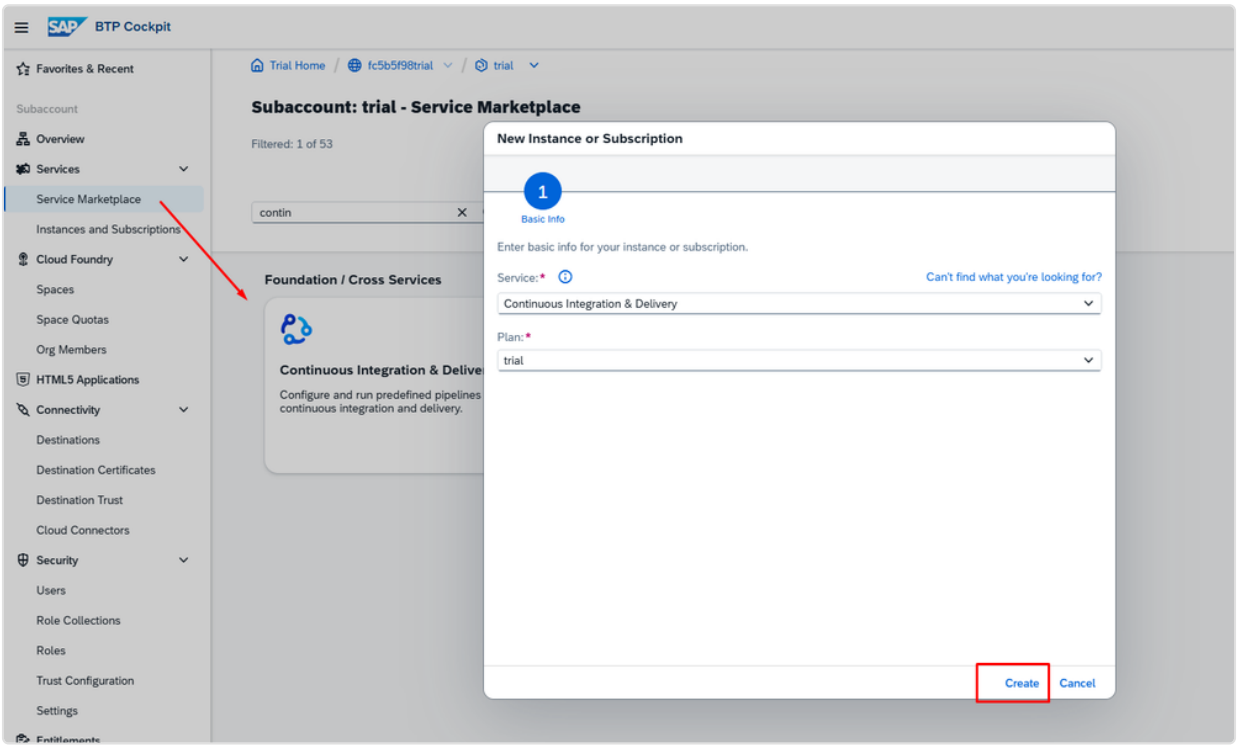


Figure: SAP BTP Service Marketplace CI/CD tile

Once the subscription is confirmed, open Instances and Subscriptions to verify that Continuous Integration & Delivery shows as Subscribed.

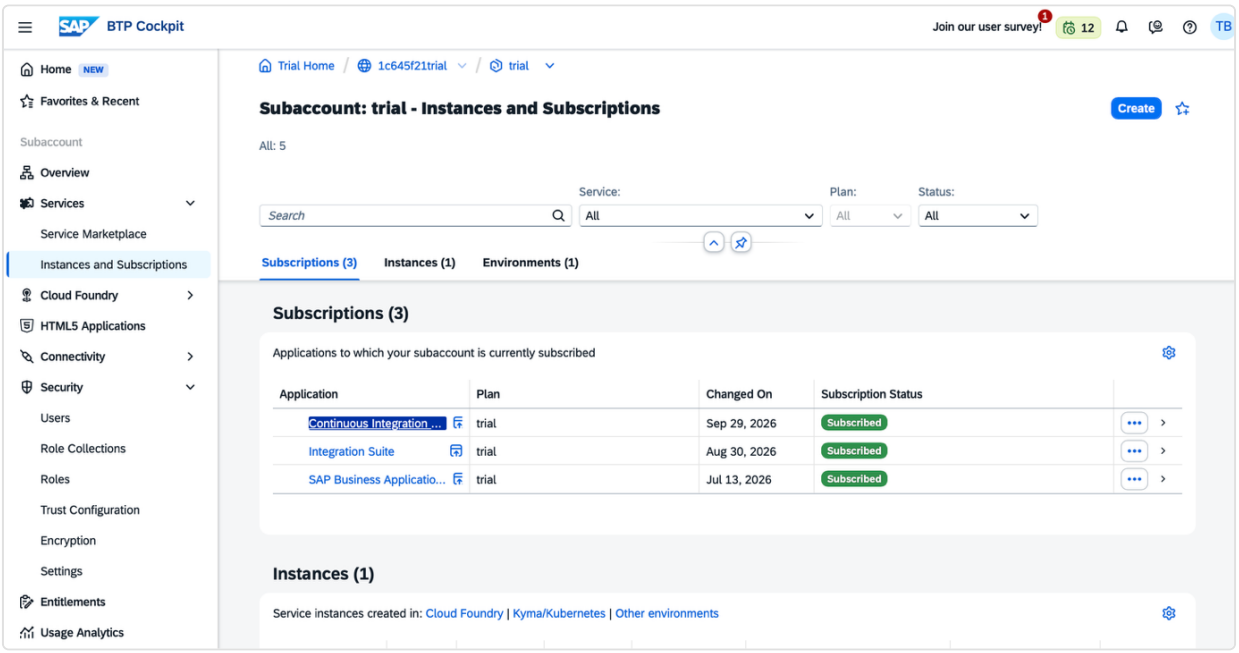


Figure: BTP subscriptions list showing CI/CD as Subscribed

Assign Required Role Collections

Navigate to **Security** → **Users** and select your user. Assign both the **CICD Service Administrator** and **CICD Service Developer** role collections. The Administrator role is required to configure credentials, repositories, and jobs. The Developer role is required to trigger and monitor pipeline runs.

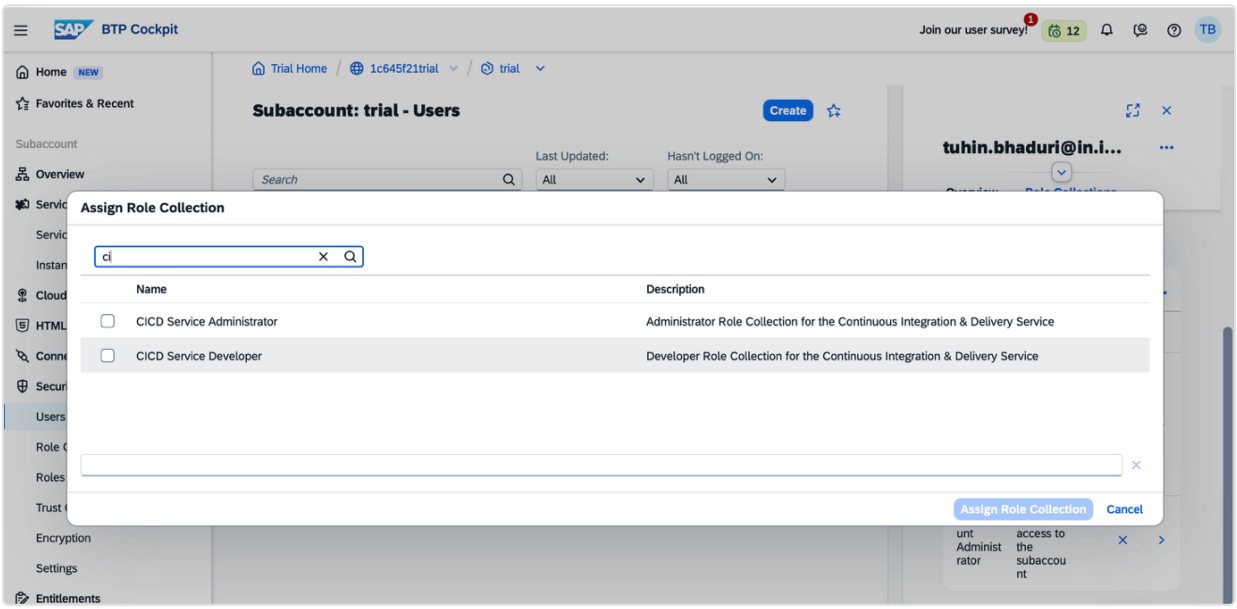


Figure: BTP Assign Role Collections dialog with CICD roles selected

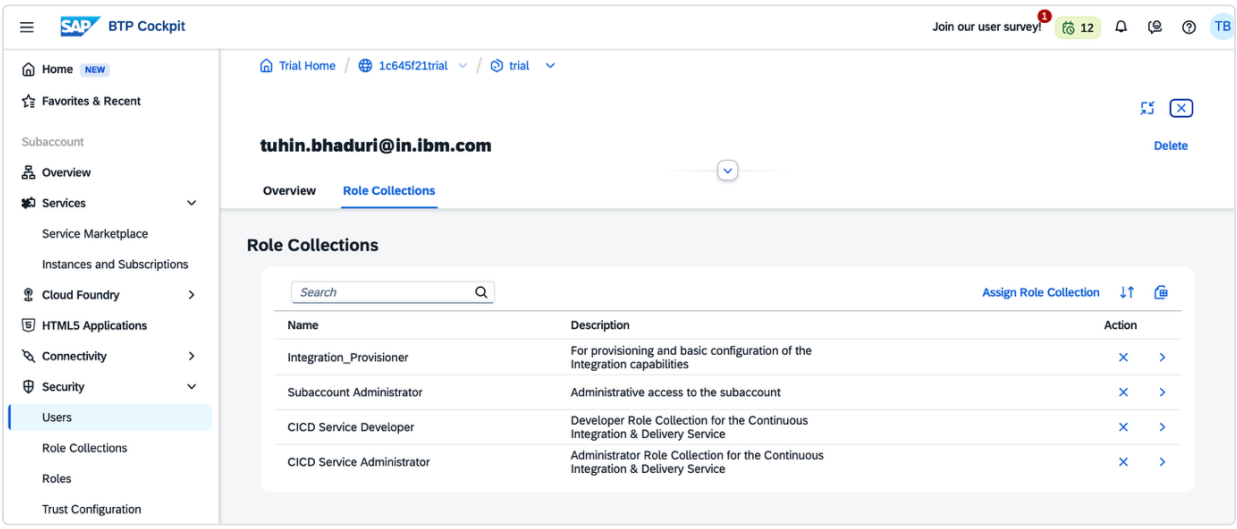


Figure: BTP user role collections confirming CICD roles

> **Important:** After assigning the role collections, sign out of BTP and sign back in before opening the Continuous Integration & Delivery application. The updated permissions are not reflected in the active session until the user re-authenticates. This is a common troubleshooting step that saves time.

Step 4: Configure Credentials in CI/CD

The CI/CD pipeline needs credentials for two systems: GitHub (to pull the source code) and SAP BTP Cloud Foundry (to deploy the application). Both credentials are created in the Credentials tab of the Continuous Integration & Delivery application.

GitHub Credential

Open the Credentials tab in CI/CD and create a new credential of type Basic Authentication. Use your GitHub username and the PAT created in Step 1 as the password. Name it clearly — for example, `githubcred` — so it can be identified when configuring the repository in the next step.

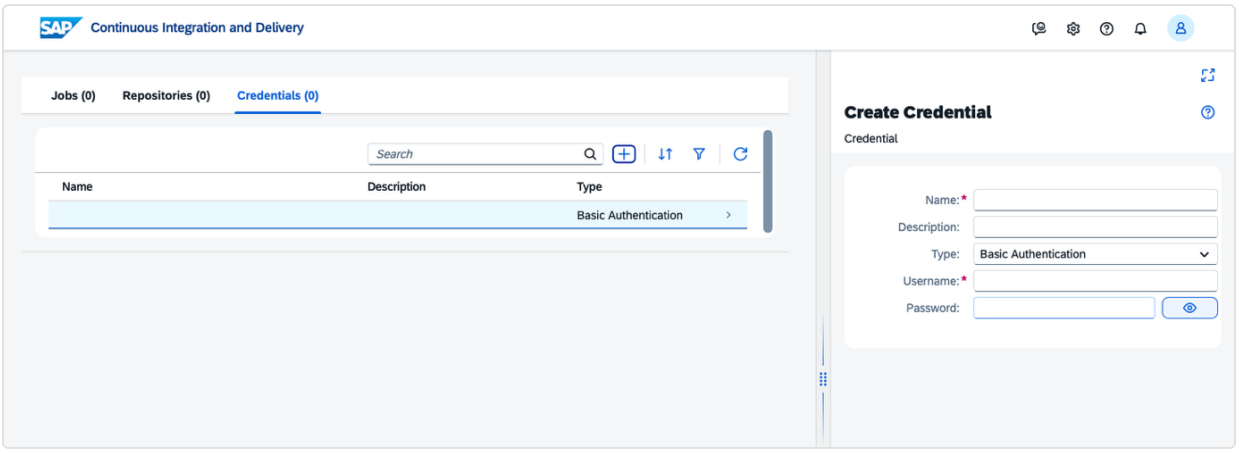


Figure: SAP CI/CD Create Credential panel with Basic Authentication

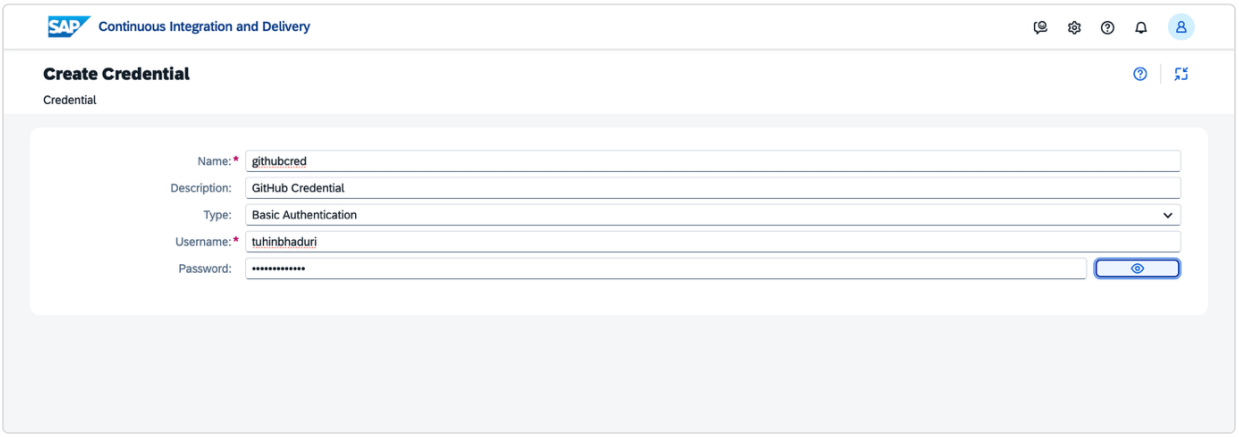


Figure: SAP CI/CD githubcred credential with Basic Authentication

SAP BTP Credential

Create a second credential for the SAP BTP environment. This credential is used by the deployment stage to authenticate with the target Cloud Foundry environment. Use your BTP email address as the username and your BTP password as the credential value.

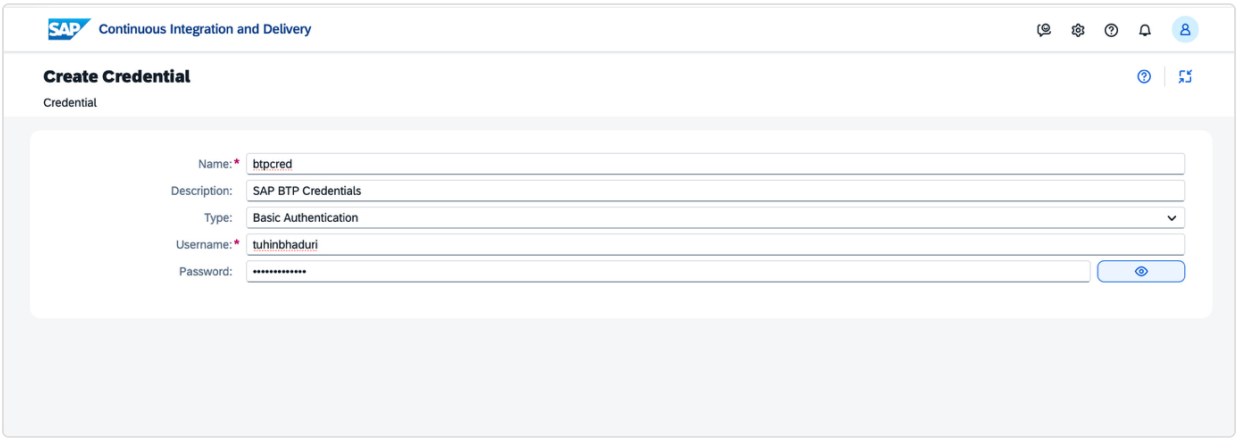


Figure: SAP CI/CD btpcred credential with Basic Authentication

Step 5: Add the GitHub Repository

Now register the GitHub repository with CI/CD. Navigate to the Repositories tab and add a new repository. Provide the repository name, the HTTPS clone URL from GitHub, and select the GitHub credential created in the previous step. The clone URL is available from the Code dropdown in the GitHub repository.

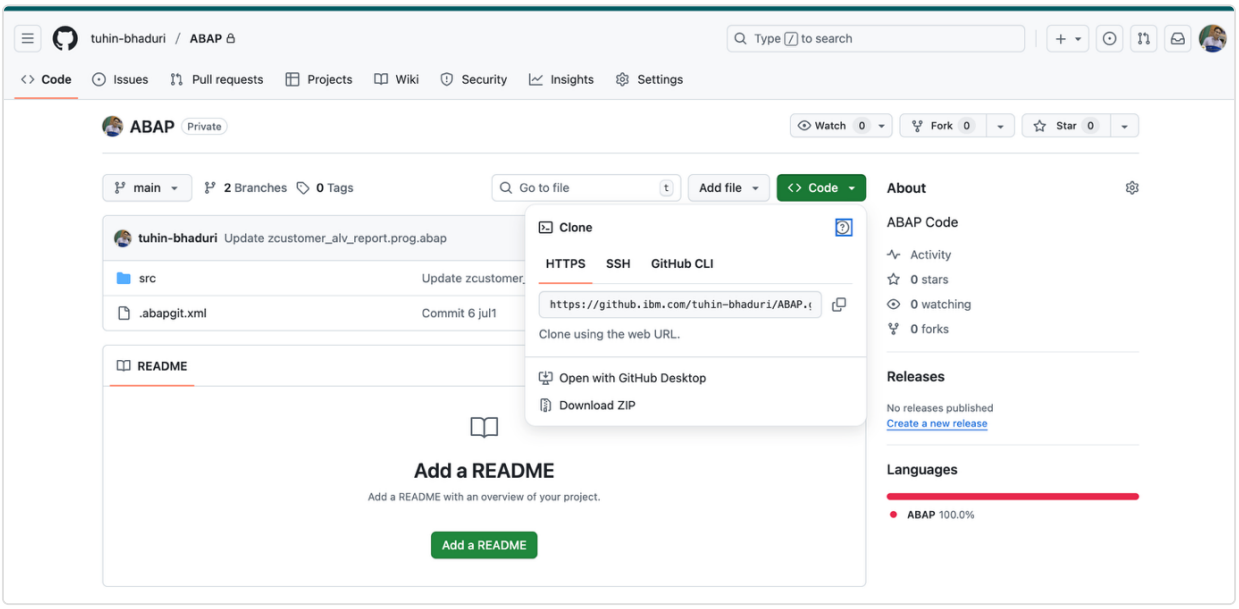


Figure: GitHub repository Code dropdown showing the HTTPS clone URL

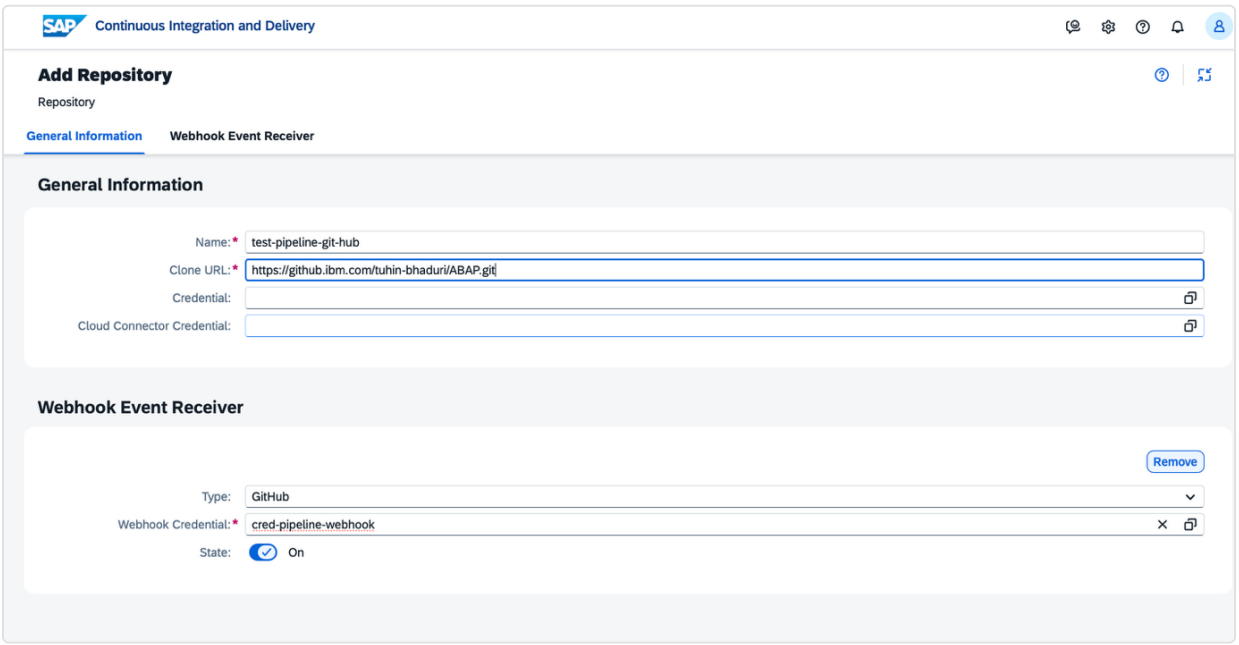


Figure: SAP CI/CD Add Repository panel with webhook credential

When the repository is saved, the CI/CD service generates the Webhook Data — including the Payload URL, Content type, and the secret reference — needed to configure the GitHub webhook in the next step.

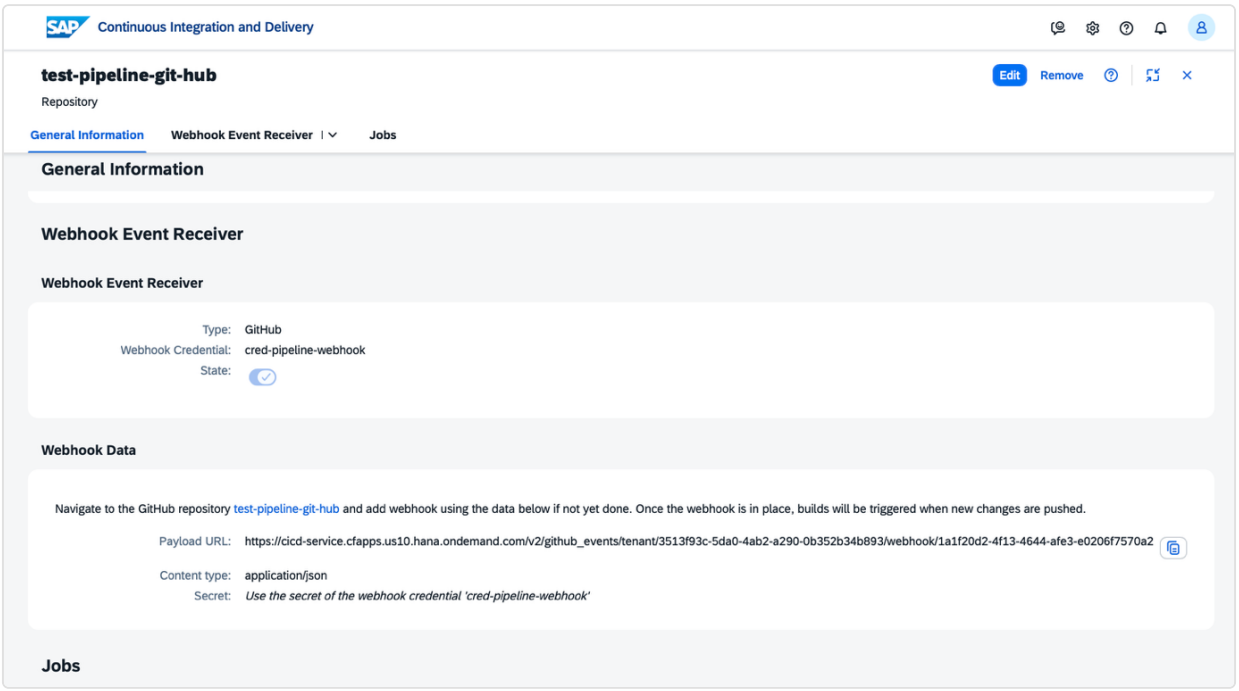


Figure: SAP CI/CD repository detail with the webhook Payload URL

Step 6: Configure the GitHub Webhook

The webhook is the mechanism that notifies SAP CI/CD when a push event occurs in the GitHub repository. The configuration has two sides: a Webhook Event Receiver in CI/CD and a Webhook Sender in GitHub. Both must be configured with the same secret for the connection to work.

Create the Webhook Credential in CI/CD

While adding the repository in CI/CD, open the Webhook Event Receiver tab. Click the credential search help and choose Create Credential. Give it a name (for example, `cred-pipeline-webhook`), and click Generate to create a random secret. Copy the secret immediately — it is shown only once.

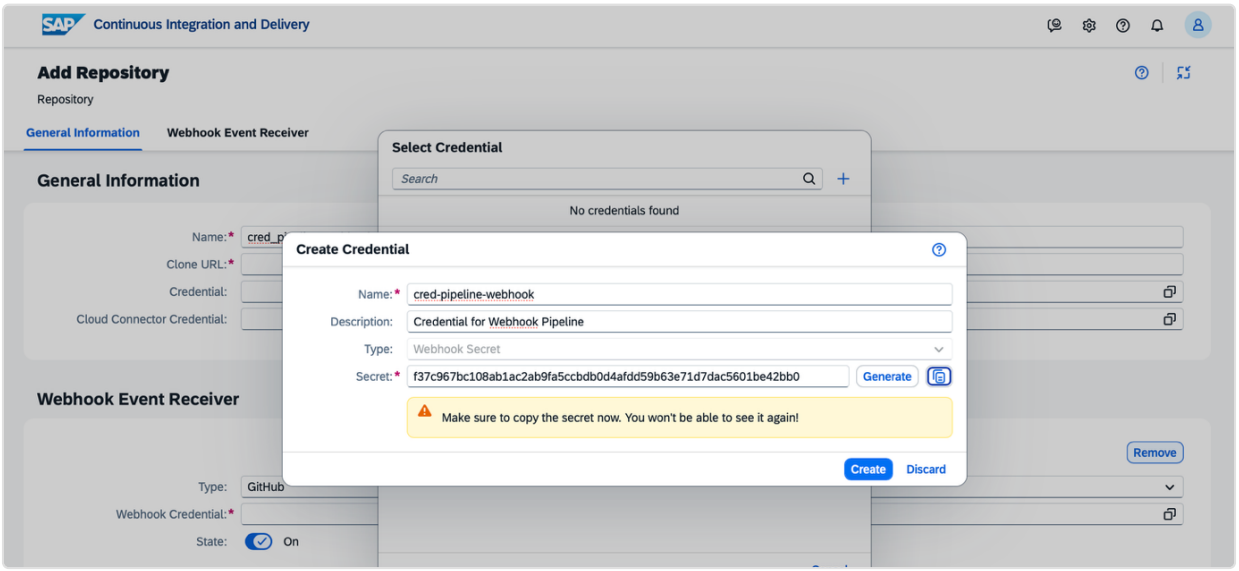


Figure: SAP CI/CD Create Credential dialog for the webhook secret

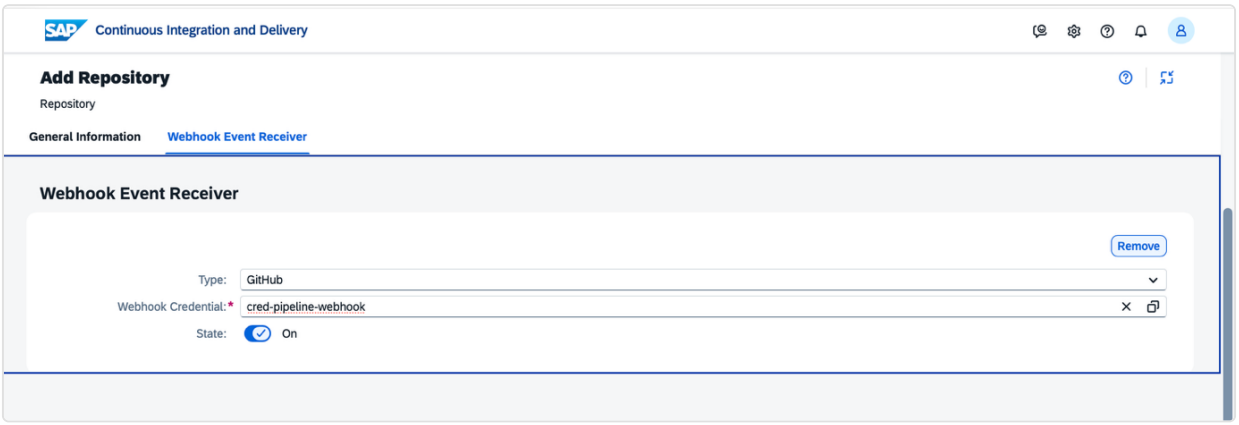


Figure: SAP CI/CD Webhook Event Receiver tab with State enabled

Create the Webhook in GitHub

In the GitHub repository, navigate to **Settings** → **Hooks** → **Add webhook**. Paste the Payload URL from the SAP CI/CD repository detail page, enter the secret copied in the previous step, and set Content type to `application/json`. Trigger the webhook on push events only.

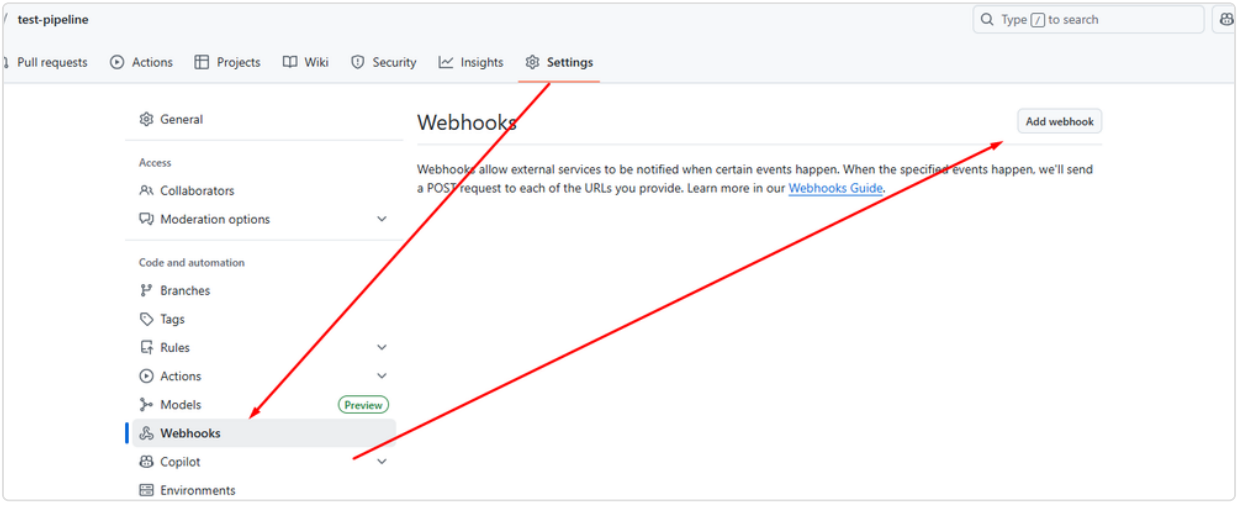


Figure: GitHub Settings Webhooks page with Add webhook button

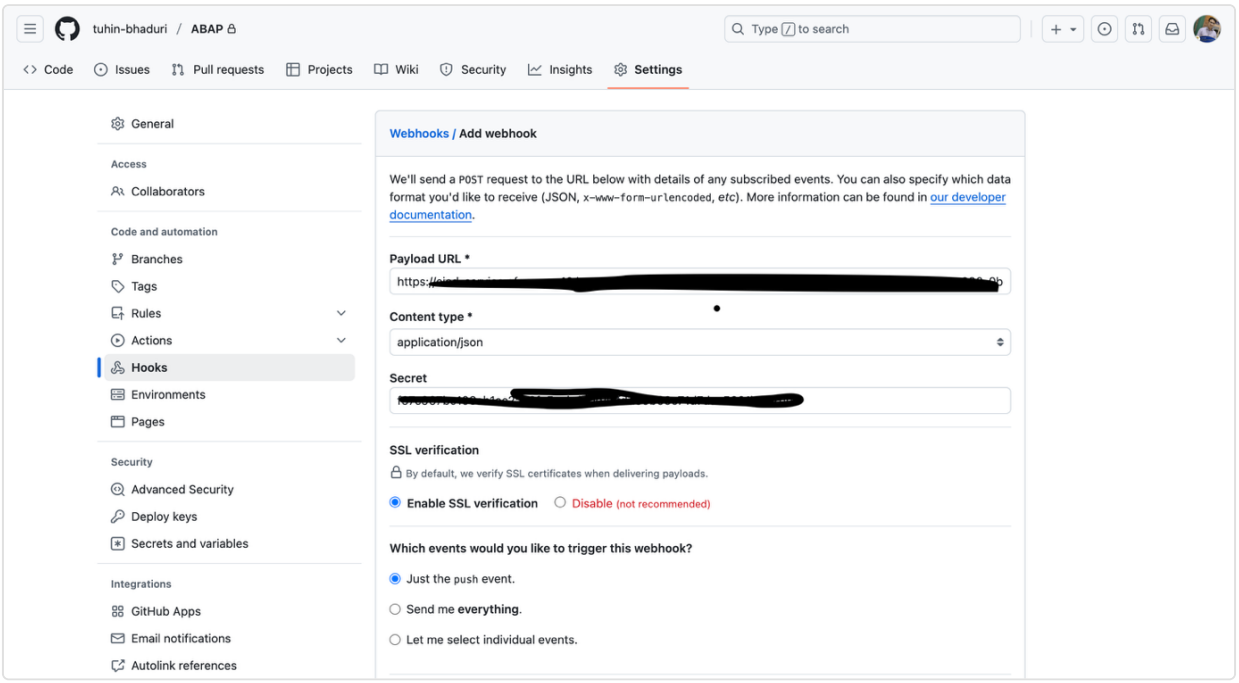


Figure: GitHub Add Webhook form configured for the CI/CD endpoint

The communication flow once the webhook is active is: GitHub Repository → (push event) → GitHub Webhook Sender → CI/CD Webhook Receiver → Pipeline Trigger. Any push to the configured branch triggers the full CI/CD run.

Step 7: Create the CI/CD Job

Navigate to the Jobs tab in CI/CD and create a new job. Provide a job name, select the repository registered in Step 5, specify the branch to monitor (typically `main`), and choose the Cloud Foundry Environment pipeline. Verify the job state is set to On.

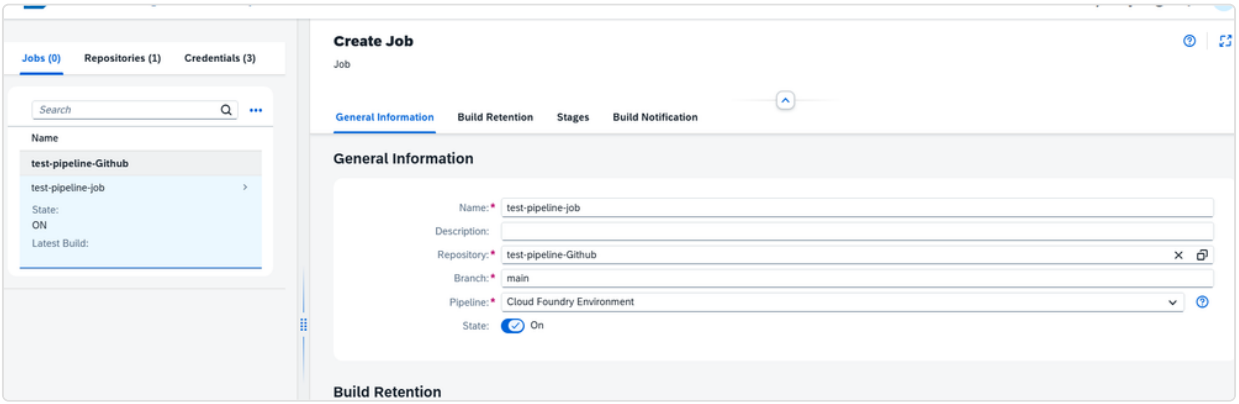


Figure: SAP CI/CD Create Job panel

Step 8: Configure Cloud Foundry Deployment

Within the job configuration, navigate to **Stages** → **Release** and enable the Deploy to Cloud Foundry Space stage. The three pieces of information required are the API Endpoint, Organization Name, and Space. All three are available from the Cloud Foundry Environment overview in the BTP Cockpit.

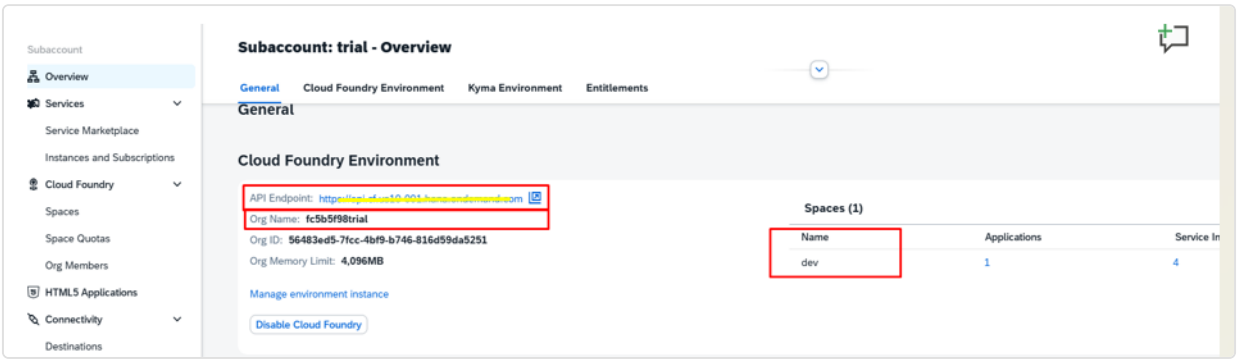


Figure: SAP BTP Cockpit Cloud Foundry environment overview

Enter those values into the deployment configuration and select the SAP BTP credential (`btpcred`) created in Step 4. The pipeline now knows where to deploy the application and how to authenticate with the Cloud Foundry environment.

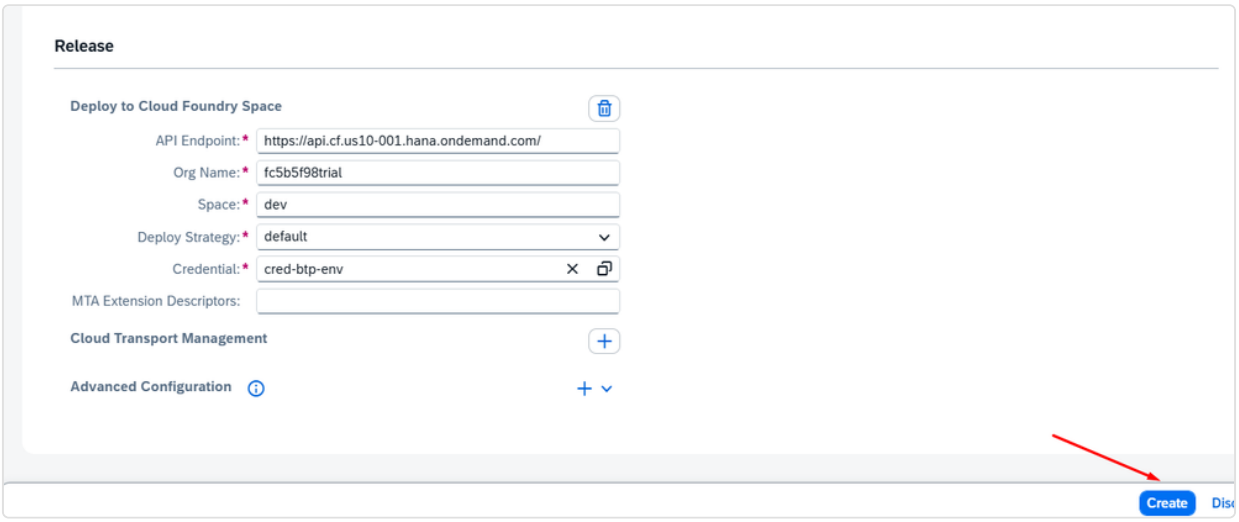


Figure: SAP CI/CD Release stage with Cloud Foundry deployment settings

Step 9: Configure Build Retention and Notifications

Review the remaining job tabs before creating the job. Build Retention controls how many historical build logs CI/CD keeps — for a trial environment, a retention of 10 builds is sufficient. Pipeline Stages allows specific stages (Clone, Build, Deploy) to be enabled or disabled. Build Notification can be configured to send an email when a run succeeds or fails. Once the configuration looks correct, click Create to save the job.

Step 10: Test the End-to-End Flow

With all configuration in place, the pipeline is ready to test. The test validates the full chain from IBM Bob commit to BTP deployment.

Make a small change to the CAP/ABAP application source inside IBM Bob — even a comment update is sufficient. Ask Bob to stage and commit the change, then push it to the GitHub repository. As soon as the push reaches GitHub, the webhook fires and CI/CD picks up the event.

Open the GitHub repository and navigate to **Settings** → **Webhooks**. The most recent delivery should show a green tick and the message "Last delivery was successful." This confirms that GitHub successfully notified CI/CD.

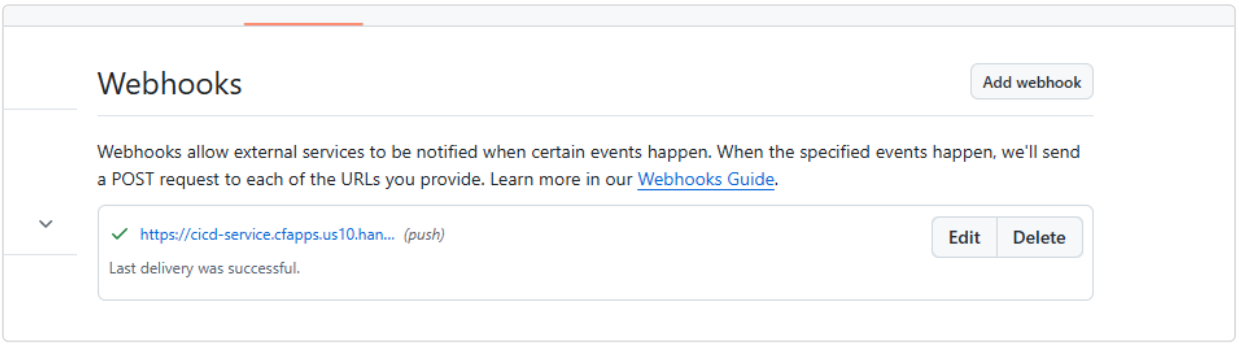


Figure: GitHub webhook with a successful last delivery

Open the SAP CI/CD application and check the job log. The pipeline should show stages: Clone → Build → Deploy. A successful run ends with the application deployed and running in the configured BTP Cloud Foundry space.

> **Note:** If the pipeline does not trigger after the commit, check these four areas in order:
>
> 1. **GitHub Webhook**: verify the Payload URL, secret, and content type, and inspect the delivery status in GitHub Settings → Webhooks
> 2. **CI/CD Repository**: verify the clone URL, the GitHub credential, and the webhook event receiver configuration
> 3. **Cloud Foundry Deployment**: verify the API endpoint, organization name, space, and BTP credential
> 4. **Application**: verify that `mta.yaml` is present, dependencies are declared, required BTP services are available, and a manual deployment completes successfully before expecting the automated pipeline to work.

The complete end-to-end delivery chain, once configured, runs without any developer intervention beyond the initial commit:

```
IBM Bob (code change in IDE)
│
▼  git push (via IBM Bob natural language)
IBM GitHub Repository
│
▼  Webhook (push event → JSON payload)
SAP CI/CD Service
│
▼  Pipeline: Clone → Build → Deploy
SAP BTP Cloud Foundry
│
▼
Application running
```

Quick Reference Card

PR Creation Flow

Table: Quick reference: context mentions, slash commands, and how to use them

Context / Command	How to use	What happens
@git-changes	Type in chat before a commit or review	Bob loads full git status + diff into context
@	Type in chat to reference a past commit	Bob loads message, author, date, and diff
/create-pr	Type as a standalone message in chat	Triggers interactive PR creation flow

Complete Command Reference

Table: Without and with IBM Bob — the impact on Git workflow discipline

Without IBM Bob	With IBM Bob
Manually write commit messages — often terse or omitting the change request reference	Bob generates conventional commit messages from the actual diff, with CHG numbers embedded automatically
Switch to terminal for git init, checkout, add, commit, push	All Git operations are available through natural language in the IDE — no terminal context-switching
Write the PR description from scratch, or leave it empty	Bob auto-generates the PR title and description from branch name, commit history, and full diff
Run linting manually — or skip it when under time pressure	Bob runs lint checks and applies fixes before the commit or PR is raised
Disconnect between Git history and ServiceNow change records	Bob embeds CHG numbers in every commit message; CI pipelines verify the trace automatically

Summary and Key Takeaways

GitHub is the version-control backbone for AI-assisted SAP delivery — not just a repository, but the platform where requirements meet code, review, and automation. IBM Bob makes GitHub a genuine productivity multiplier for SAP developers, not by adding complexity but by removing friction from the operations that developers already need to perform.

The two integration pathways — the GitHub MCP Server for cross-repository and programmatic operations, and the built-in Git workflow for day-to-day development — serve different purposes and complement each other. Both are available simultaneously. The right choice depends on the task.

The CI/CD pipeline examples in this section demonstrate what a fully automated SAP delivery chain looks like end-to-end: a natural language commit from inside IBM Bob becomes a deployed BTP application without further manual intervention. The traceability chain — from ServiceNow change request to branch name to commit message to CI check to deployment — is established automatically by the patterns Bob follows, not enforced by process overhead that teams work around under pressure.

IBM Bob and Tricentis: AI-Assisted SAP Test Automation

> **Attention:** All examples, scenarios, test case names, repository structures, and system configurations shown in this section are for demonstration purposes only. They do not represent standard projects, templates, or recommended implementations supplied by IBM, SAP, Tricentis, or any other vendor.

Test automation is one of the most time-consuming disciplines in SAP delivery. Building a reliable regression suite, keeping it aligned with an evolving application, and maintaining the test scripts when changes break them — all of this traditionally requires a significant and sustained manual effort. Tricentis changes that picture dramatically on the automation side. IBM Bob changes it on the intelligence side. Together, they create a test automation capability that can keep pace with the speed of AI-assisted development.

Tricentis is the market-leading platform for end-to-end SAP test automation. Its flagship product, Tosca, supports SAP GUI, SAP Fiori, SAP S/4HANA, SAP ECC, SAP BTP, and a wide range of browser-based application interfaces. Beyond functional testing, Tricentis covers regression testing, performance testing, and volume testing — making it a single platform capable of serving every test phase in the SAP delivery lifecycle.

IBM Bob connects to Tricentis through the ToscaCloud MCP Server, an official Model Context Protocol integration that gives Bob programmatic access to the Tricentis platform. When Bob is connected, it can read test issues from Jira or ServiceNow, trace them to specific test scripts in Tricentis, apply corrections, update test automation assets, and generate new test cases from requirements — all without the test engineer leaving the IDE.

Tricentis in the SAP Testing Landscape

Most SAP programs face the same test automation challenge: building reliable automated tests takes significant effort upfront, and maintaining them as the application evolves requires a sustained investment that teams rarely budget for. The result is that regression suites either never get fully built, or they get built once and then gradually fall out of sync with the application until they are no longer trusted. Tricentis addresses this challenge by providing a platform designed specifically for enterprise application testing at scale.

What Tricentis Does

Tricentis Tosca is a test automation suite that handles the full test lifecycle — test design, automation script creation, test execution, reporting, and deployment — within a single platform. Unlike general-purpose automation frameworks that require significant scripting expertise, Tosca uses a model-based approach that makes test creation accessible to business analysts, functional testers, and developers alike.

For SAP environments specifically, Tricentis provides built-in connectors for every major SAP interface. SAP GUI transactions, SAP Fiori applications running in a browser, and SAP BTP services can all be tested through the same Tosca platform without requiring separate tools or custom integrations. The platform also supports:

Table: Tricentis test types and their roles in SAP delivery

Test Type	What it validates
Functional testing	Validates that individual SAP transactions and processes produce the correct output for a given set of inputs
Regression testing	Confirms that changes to the application have not broken existing functionality — the primary use case for automated SAP testing
Performance testing	Measures system response times and throughput under defined load conditions (often combined with NeoLoad)
Volume testing	Verifies that the application performs correctly with large data sets representative of production volumes
End-to-end testing	Validates complete business process flows — for example, Order to Cash or Procure to Pay — across multiple SAP systems

Where Tricentis Fits in the SAP SDLC

In a traditional SAP delivery program, automated testing happens late — often only during the system integration test (SIT) or user acceptance test (UAT) phase, when defects are expensive to fix. Tricentis changes this dynamic by

making it practical to automate testing continuously throughout the development cycle, from unit testing of individual ABAP programs or Fiori components through to end-to-end regression suite execution before every production release.

The business case for this investment is strongest in two scenarios: first, during S/4HANA migrations, where the application footprint changes substantially and a large regression suite is the only reliable way to validate that critical processes still work; second, in production support, where quarterly SAP patches and regular enhancement releases require rapid regression validation that cannot be done manually within the available time window.

> **Tip:** The combination of Tricentis and IBM Bob is particularly valuable in S/4HANA migration programs, where the test automation suite itself needs to be built or rebuilt alongside the new system. Bob can generate test cases from functional specifications, create descriptive programming modules, and maintain the suite as development progresses — compressing what would traditionally be months of test preparation into weeks.

The IBM Bob and Tricentis Integration Architecture

The integration between IBM Bob and Tricentis operates across three distinct but connected layers. Understanding these layers is important for both architects designing the integration and for developers and test engineers working with it day-to-day.

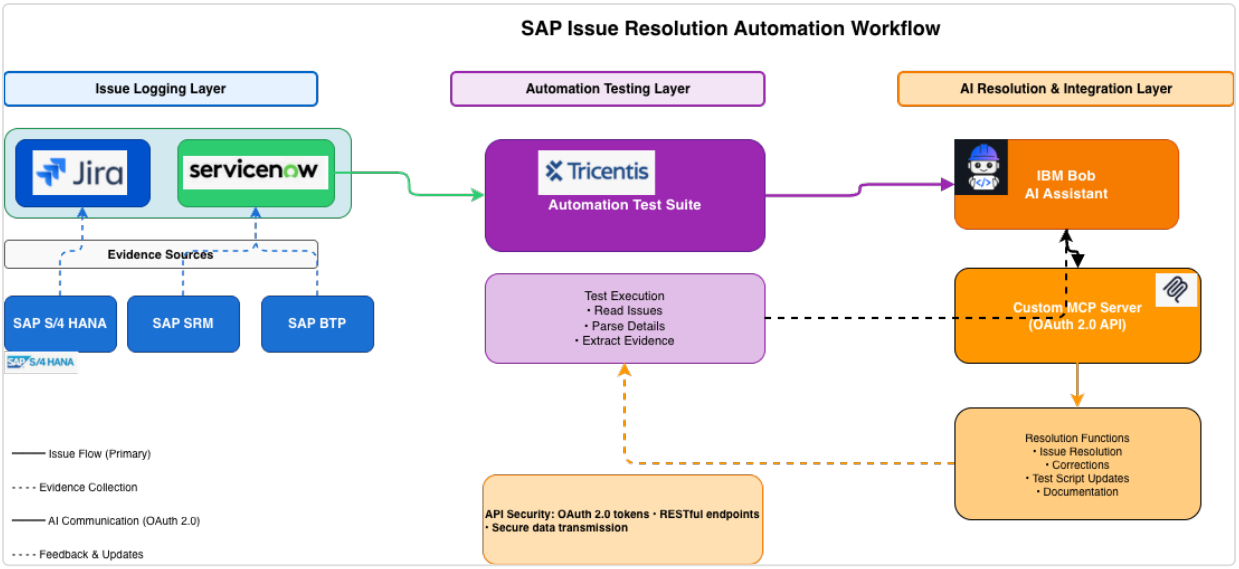


Figure: SAP issue resolution automation workflow

Layer 1: Issue Logging

The first layer is the issue management system where test failures and defects are recorded. In most SAP delivery programs, this is either Jira or ServiceNow — both of which have dedicated IBM Bob integrations described in the IBM Bob and ServiceNow: Integrating the AI Developer Assistant with Enterprise Change Management and IBM Bob and GitHub: AI-Assisted Version Control and CI/CD for SAP Delivery sections. When a Tricentis test run identifies a failure, the defect is raised in the issue management system with the relevant test evidence: the test case name, the failed step, the expected versus actual result, and any supporting screenshots or log extracts.

The SAP systems that generate test evidence feed into this layer too. Issues arising from SAP S/4HANA, SAP SRM, or SAP BTP processing failures are captured in the same Jira or ServiceNow instance, creating a single source of truth for all defects regardless of origin.

Layer 2: Automation Testing

The second layer is the Tricentis Automation Test Suite itself. This layer is responsible for executing test scenarios, collecting test results, and providing the evidence that feeds the issue resolution process. When IBM Bob interacts with this layer through the MCP connection, it can read the full context of a test failure — the test module, the test steps, the assertion that failed, and the data used in the test run — without any manual extraction or copy-paste by the tester.

This layer also includes the API security boundary: all communication between IBM Bob and the Tricentis platform is secured with OAuth 2.0 tokens and RESTful endpoints. This means the integration is compatible with enterprise security requirements and does not require any network exceptions or special access configurations beyond the standard OAuth flow.

Layer 3: AI Resolution and Integration

The third layer is where IBM Bob operates. Once Bob receives the issue context — either directly from Jira or ServiceNow, or via the Tricentis test evidence — it uses the Custom MCP Server (the ToscaCloud MCP Server) to connect to the Tricentis platform and perform the resolution. The resolution functions available to Bob through this layer include:

Table: IBM Bob resolution functions available through the ToscaCloud MCP Server

Resolution Function	What Bob does
Issue resolution	Analyzes the failing test step and root cause, then proposes and applies the correction
Script corrections	Updates the Tricentis test script directly to fix the failing assertion or test step
Test script updates	Modifies test data, selectors, or process steps in response to application changes
Documentation	Generates updated test case documentation reflecting the corrected state of the test script

Connecting IBM Bob to Tricentis Tosca Cloud

The connection between IBM Bob and Tricentis is established through the ToscaCloud MCP Server — an MCP integration available in IBM Bob's MCP Marketplace. Once connected, the server appears in Bob's MCP settings alongside any other configured integrations, and Bob can use its tools in any conversation where Tricentis context is relevant.

The ToscaCloud MCP Server

The ToscaCloud MCP Server exposes the Tricentis Tosca Cloud platform as a set of callable tools within IBM Bob. The authentication mechanism is OAuth 2.0, which means access is governed by the same token-based security model used by the Tricentis REST API. The connection is configured once and then available globally or per workspace, depending on the scope chosen during setup.

Alongside ToscaCloudMcpServer, the Tricentis integration also includes the `ai-workspace` MCP server, which provides access to the Tricentis AI Workspace — a purpose-built environment where AI agents can run test workflows, generate test cases, and manage test execution results. Both servers can be active simultaneously and complement each other: ToscaCloudMcpServer handles script-level interactions, while `ai-workspace` handles higher-level AI-driven test generation and run management.

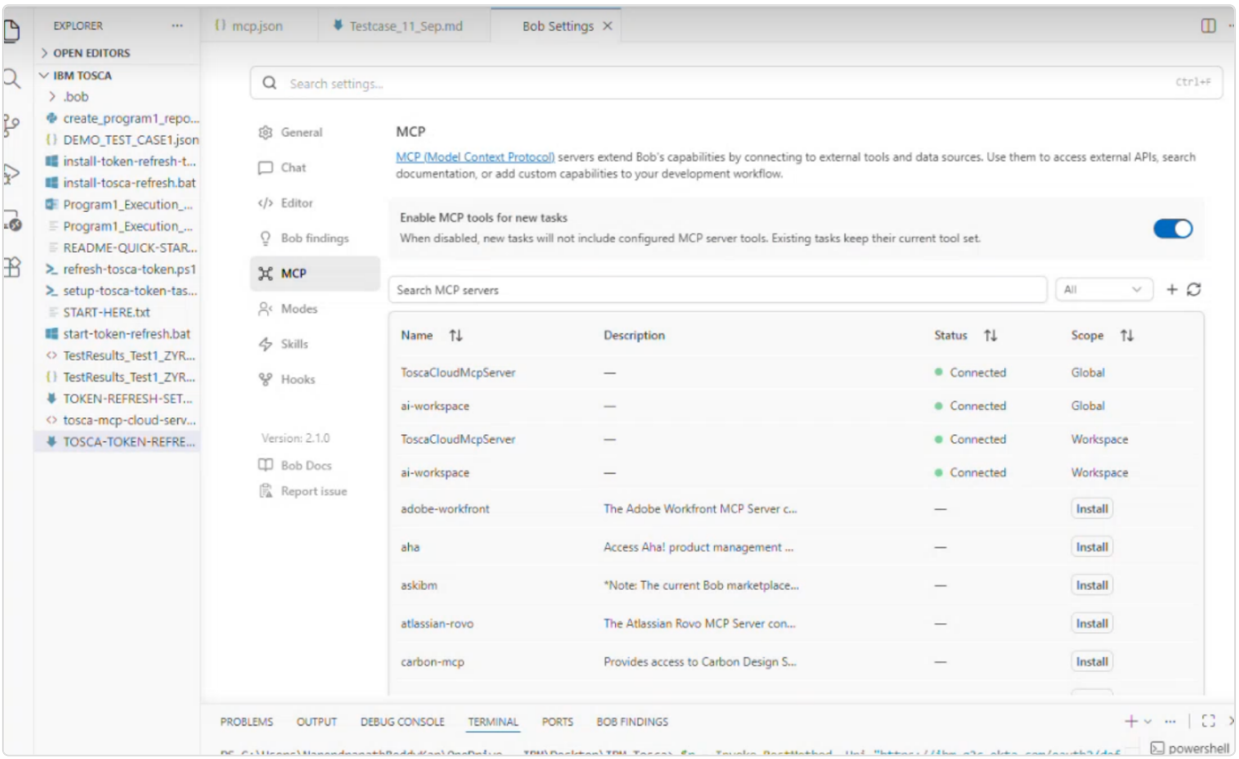


Figure: IBM Bob MCP settings with Tricentis servers connected

> **Note:** In the screenshot above, both ToscaCloudMcpServer and ai-workspace appear twice — once at Global scope and once at Workspace scope. Global scope makes the MCP server available in all Bob conversations. Workspace scope restricts it to the current project folder. For a team working on a specific SAP program, Workspace scope is typically the right choice, as it avoids inadvertently using Tricentis tools in unrelated conversations.

mcp.json Configuration

The ToscaCloud MCP Server is configured in the `.bob/mcp.json` file in the project directory. Like all Bob MCP integrations, the authentication token is referenced as an environment variable — never stored directly in the configuration file.

```
"ToscaCloudMcpServer": {
"command": "npx",
"args": ["-y", "@tricentis/tosca-cloud-mcp-server"],
"env": {
"TOSCA_CLIENT_ID":     "${env:TOSCA_CLIENT_ID}",
"TOSCA_CLIENT_SECRET": "${env:TOSCA_CLIENT_SECRET}",
"TOSCA_CLOUD_URL":     "${env:TOSCA_CLOUD_URL}"
}
},
"ai-workspace": {
"command": "npx",
"args": ["-y", "@tricentis/ai-workspace-mcp-server"],
"env": {
"AI_WORKSPACE_URL":    "${env:AI_WORKSPACE_URL}",
"AI_WORKSPACE_TOKEN":  "${env:AI_WORKSPACE_TOKEN}"
}
}
```

> **Important:** Store all Tricentis credentials as shell environment variables — never in the `.bob/mcp.json` file itself. The `mcp.json` file is typically committed to source control, and any credentials embedded in it would be exposed. Use the `${env:VARIABLE_NAME}` syntax consistently across all MCP integrations.

AI-Assisted Test Case Generation

One of the most immediate productivity benefits of connecting IBM Bob to the Tricentis AI Workspace is the ability to generate comprehensive test cases from requirements or functional specifications. Instead of a test engineer reading a requirement document and manually writing test cases one by one, Bob reads the requirements and generates a complete, numbered test case set — with source requirement traceability, purpose, expected outcome, test data, preconditions, and step-by-step test steps — ready for review and import into Tricentis.

The Tricentis AI Workspace

The Tricentis AI Workspace is a purpose-built environment within the Tricentis platform where AI agents can execute multi-step workflows. A workflow might receive a set of requirements, generate test cases, validate them against a testing standard, and output a formatted test case document — all in a single automated run. The AI Workspace is accessible through IBM Bob via the `ai-workspace` MCP server, which means all of this can be triggered from a natural language prompt inside the IDE.

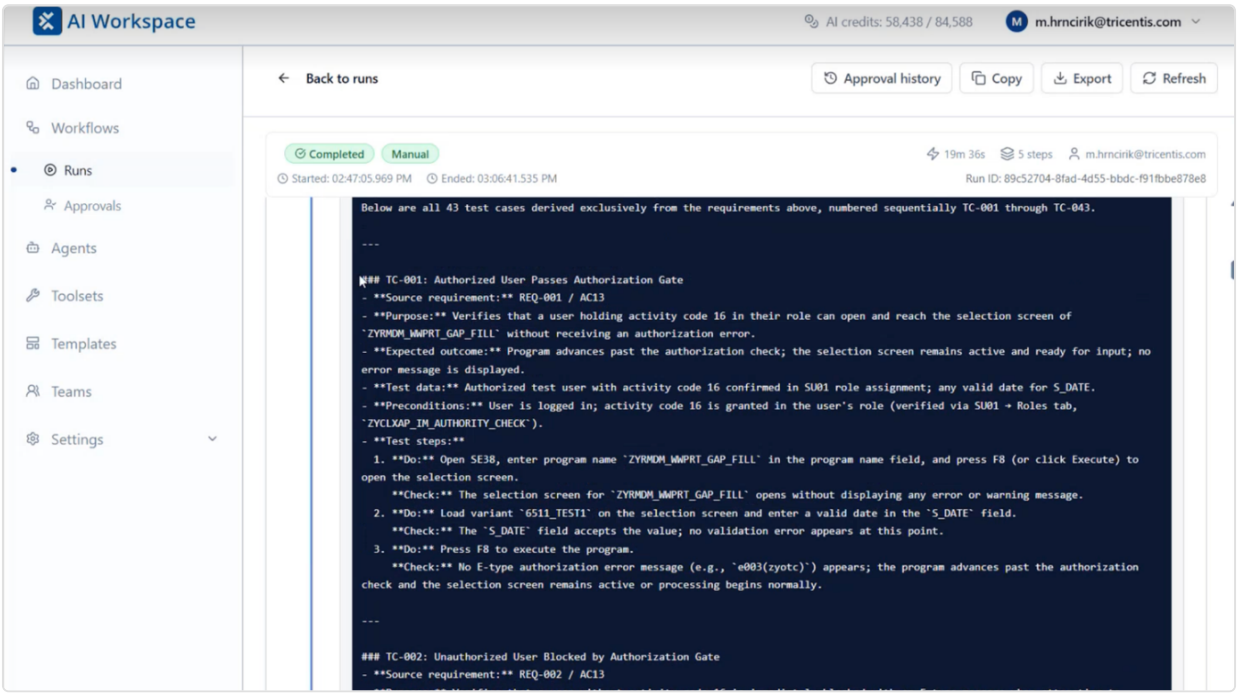


Figure: Tricentis AI Workspace run with 43 generated test cases

The screenshot above shows a completed AI Workspace run that generated 43 test cases from a set of requirements. Each test case follows a consistent structure: Source Requirement, Purpose, Expected Outcome, Test Data, Preconditions, and Test Steps. This level of consistency and completeness, produced automatically from requirements, eliminates the variability and incompleteness that typically characterize manually written test cases.

An example of a generated test case (TC-001) from the run shown above:

```
TC-001: Authorized User Passes Authorization Gate

Source requirement:  REQ-001 / AC13

Purpose: Verifies that a user holding activity code 16 in their role can open the selection screen of ZYRM

Expected outcome: Program advances past the authorization check; selection screen remains active and ready

Test data: Authorized test user with activity code 16 confirmed in SU01 role assignment; any valid date fo

Preconditions: User is logged in; activity code 16 is granted in the user's role (verified via SU01 → Role


Test steps:

1. Do: Open SE38, enter ZYRMDM_WWPRT_GAP_FILL, press F8 to execute.

Check: Selection screen opens without any error or warning message.

2. Do: Load variant 6511_TEST1, enter a valid date in S_DATE.

Check: S_DATE accepts the value; no validation error appears.

3. Do: Press F8 to execute the program.

Check: No E-type authorization error (e003/zyotc) appears; program advances past the authorization check n
```

> **Tip:** The AI Workspace run shown in the screenshot above took under 20 minutes (the run log shows 19m 36s elapsed) to generate 43 test cases from requirements. Producing the same set manually — with consistent structure, complete test data, and full requirement traceability — would typically take a test engineer two to three days.
>
> The value is not just speed, the generated test cases are more consistent and more complete than manually written ones, because Bob applies the same structure and level of detail to every test case in the set.

Natural Language Test Case Generation

Test case generation does not require using the AI Workspace directly. IBM Bob can generate Tricentis-compatible test cases from a natural language description of the requirement, a user story, a functional specification document, or even a description of what a transaction is supposed to do. The developer or test engineer simply describes the scenario in the Bob chat, and Bob produces a structured test case ready for import into Tricentis.

Table: Natural language prompts for IBM Bob test case generation with Tricentis

You say to Bob…	Bob produces…
"Generate test cases for the authorization check in ZYRMDM_WWPRT_GAP_FILL based on requirements REQ-001 through REQ-005"	Complete test case set with source traceability, expected outcomes, test data, preconditions, and numbered test steps for each requirement
"Review the test cases in Testcase_11_Sep.md and check for completeness against SAP testing best practices"	Gap analysis report listing missing preconditions, incomplete test data definitions, or steps that lack explicit check actions
"Create a Tosca test module for the login and navigation steps common to all the generated test cases"	Reusable Tricentis module definition that can be referenced across multiple test cases instead of duplicating the steps
"Update test case TC-003 to reflect the new field added in the latest change request"	Updated test case with the new step incorporated at the correct position, with the check action and expected value defined

Automated Issue Resolution in Tricentis Test Scripts

When a Tricentis test run fails, the traditional resolution process follows a predictable but time-consuming path: a tester reads the failure, opens the test script in Tosca, investigates the root cause, makes the correction, re-runs the affected test cases, and closes the defect. IBM Bob compresses this cycle by taking on the investigation and correction work, leaving the tester to review and approve rather than diagnose and fix.

How Bob Reads Test Failures

When a test run fails in Tricentis, the failure information — test case name, failed step, expected versus actual result, and any error messages — is surfaced to IBM Bob either directly through the ToscaCloud MCP Server or via a Jira or ServiceNow issue that the tester raises from the Tricentis failure report. In either case, Bob receives the full context needed to understand what failed and why.

The issue reaches Bob as a structured document — typically a Markdown file containing the defect description, the test evidence, and any relevant SAP error messages. Bob reads this document alongside the test script context from Tricentis and forms a diagnosis. The diagnosis is not a keyword search or a pattern match — it is a genuine root cause analysis that considers the test step logic, the SAP process being tested, and the type of failure reported.

How Bob Applies Corrections

Once Bob has identified the root cause, it uses the ToscaCloud MCP Server to apply the correction directly in the Tricentis test script. The correction might be updating a test step's expected value to reflect a changed SAP

configuration, replacing a hard-coded screen element reference with a descriptive programming call, adding a missing wait condition that causes intermittent failures, or modifying the test data used in the assertion.

After applying the correction, Bob re-runs the affected test case through the Tricentis platform and confirms that it passes. The tester receives a summary of what was changed, why, and what the test result was after the correction — enough information to review and approve the fix without needing to step through the script manually.

> **Note:** Bob's corrections are visible in the Tricentis platform in the same way as any other script change. The audit trail in Tosca shows what changed, when, and under which run context. If a correction later proves to be wrong, rolling it back follows the standard Tricentis version management process — there is no special IBM Bob undo mechanism required.

The Issue Resolution Workflow in Practice

A complete issue resolution cycle with IBM Bob and Tricentis looks like this:

```
1. Test run fails in Tricentis → Failure logged in Jira / ServiceNow with test evidence

2. IBM Bob reads the issue from Jira or ServiceNow → Gets test case name, failed step, error detail, SAP c

3. Bob connects to Tricentis via ToscaCloud MCP Server → Reads the full test script and module context

4. Bob performs root cause analysis → Identifies whether the failure is a script defect, a data issue,an a

5. Bob applies the correction to the Tricentis test script → Updates the relevant test step, assertion, or

6. Bob re-runs the corrected test case → Confirms the test now passes

7. Bob generates a correction summary → Documents root cause, change made, and test result for review

8. Tester reviews and approves → Closes the issue in Jira / ServiceNow
```

Descriptive Programming and IBM Bob

Tricentis supports two approaches to test automation: recording-based automation, where the test tool captures screen interactions and replays them, and descriptive programming, where the test script calls reusable functional modules rather than hard-coding specific screen element interactions. Descriptive programming produces more maintainable, more resilient tests — but it requires more upfront investment, because the test engineer must first identify which modules are relevant and how to call them correctly.

What Descriptive Programming Is

In descriptive programming, every test interaction is expressed as a call to a business-level function rather than as a direct screen element action. Instead of clicking on a specific button at a specific coordinate on the screen, the test calls a function named something like "ClickExecuteButton" or "EnterDateInSelectionField". These functions are defined once in a shared module library, and any test case can call them by name.

The advantage is resilience: when the SAP application changes its layout or updates a UI element, only the function definition in the module library needs to change — not every test case that uses that element. In large test suites with hundreds of test cases, this distinction between recording-based and descriptive programming can mean the difference between maintaining a suite that takes weeks to update after every release and one that takes hours.

Where IBM Bob Accelerates Descriptive Programming

The challenge with descriptive programming is discovery. A developer building a new test scenario must identify which existing modules in the library are relevant, understand the correct parameters for calling them, and determine

whether a new module needs to be created or whether an existing one can be adapted. In a mature Tricentis installation with hundreds of modules, this discovery process can take a significant portion of the overall test development effort.

IBM Bob shortens this process substantially. Because Bob has access to the Tricentis module library through the ToscaCloud MCP Server, it can search the library, identify relevant modules, explain their parameters, and suggest which combination of module calls is appropriate for a specific test scenario. A test engineer can describe a new test scenario in plain English, and Bob will return a suggested module call sequence with parameter values and the reasoning behind each choice.

Table: IBM Bob support for Tricentis descriptive programming

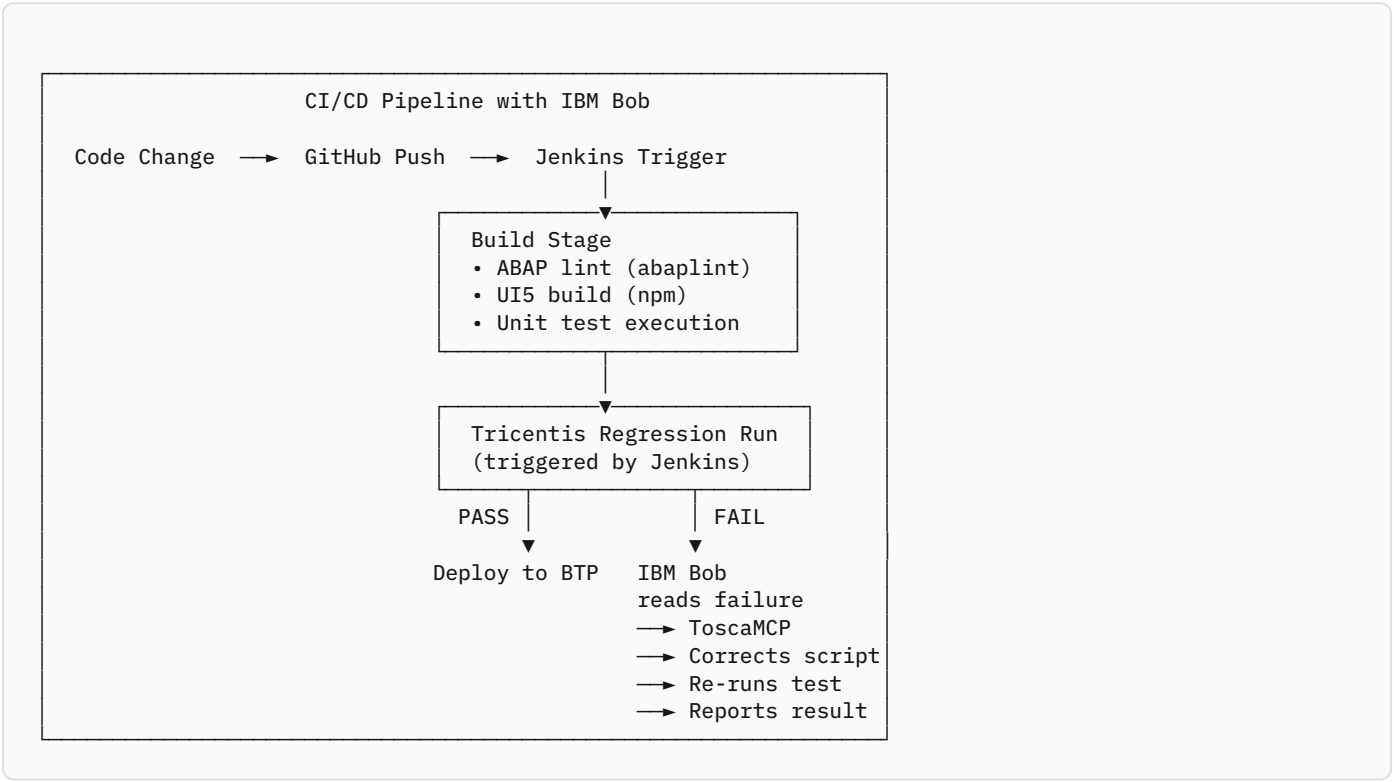
You say to Bob…	Bob does…
"I need to automate the login, navigate to MM03, search for material X, and verify the storage location field"	Searches the Tricentis module library, identifies the Login, NavigateToTransaction, SearchMaterial, and ReadFieldValue modules, and suggests the call sequence with correct parameter names
"Which modules should I use for an ABAP SE38 test that loads a variant and checks the output list?"	Returns the relevant SE38 execution modules, identifies the variant loading function, and recommends the output list validation approach based on existing modules

CI/CD Integration: IBM Bob, Tricentis, and Jenkins

The IBM Bob and Tricentis integration reaches its full potential when it is embedded in a CI/CD pipeline. In this configuration, every code change that IBM Bob pushes to GitHub triggers a Tricentis regression run automatically. If the run fails, Bob reads the failure, resolves the test script issues, and re-runs — all without human intervention. The developer's only responsibility is to review the outcome and approve the correction.

The Pipeline Architecture

The CI/CD pipeline that combines IBM Bob, Tricentis, and Jenkins follows a standard continuous delivery pattern with three additional AI-assisted steps:



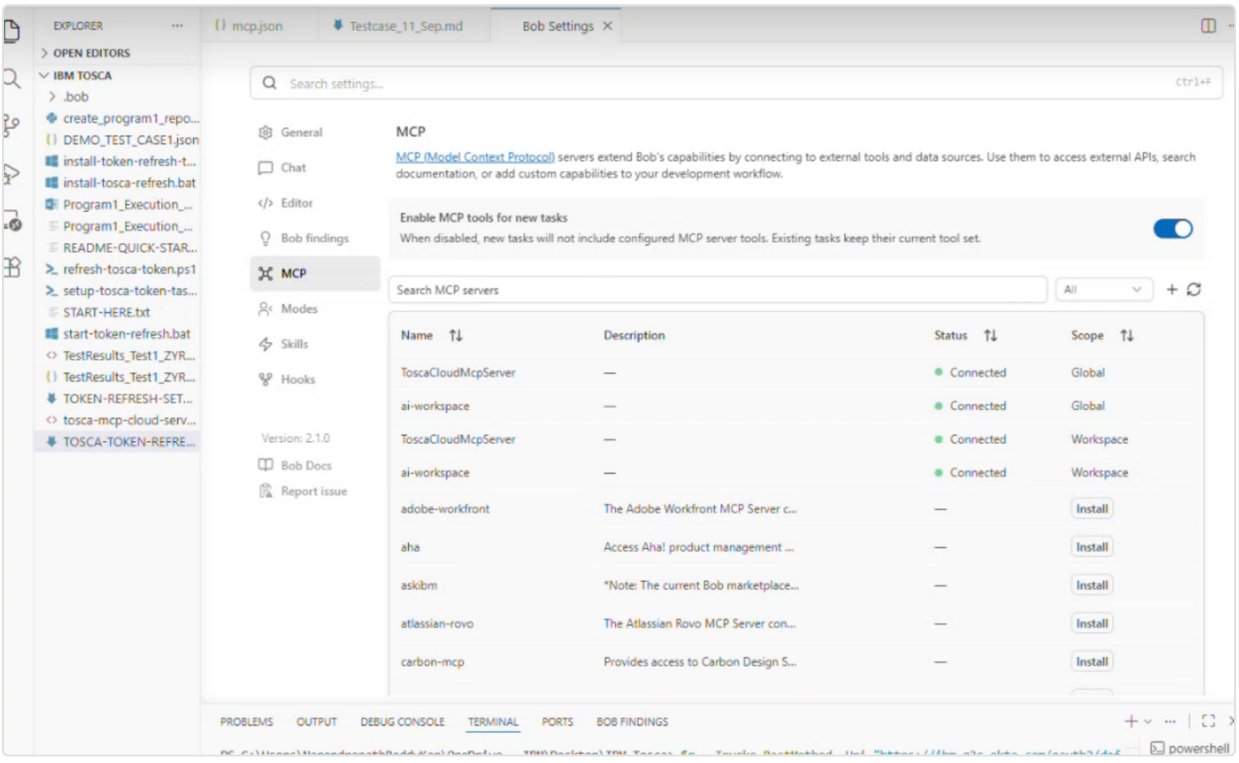


Figure: IBM Bob MCP settings showing Tricentis MCP server connections

Jenkins Integration with Tricentis

Jenkins triggers Tricentis test runs through the Tricentis CI Client or via REST API calls to the Tricentis Tosca Cloud. The Jenkins pipeline step that invokes Tricentis passes the target environment details — the SAP system URL, the

test workspace, and the test suite to execute — and waits for the run to complete. The run result (pass/fail) becomes the gate that controls whether the pipeline proceeds to the deployment stage.

```
// Jenkins pipeline step – Tricentis test execution

stage("Tricentis Regression Test") {
  steps {
    script {
      def result = sh(
        script: """
          curl -X POST ${TOSCA_CLOUD_URL}/api/v1/executions \\
            -H "Authorization: Bearer ${TOSCA_TOKEN}" \\
            -H "Content-Type: application/json" \\
            -d '{"suiteId": "${TEST_SUITE_ID}", "environment": "${TARGET_ENV}"}'
        """,
        returnStdout: true
      ).trim()

      if (result.contains('"status": "FAILED"')) {
        error "Tricentis regression run failed – IBM Bob correction workflow triggered"
      }
    }
  }
}
```

> **Tip:** When the Jenkins pipeline step reports a Tricentis failure, configure the pipeline to notify the IBM Bob webhook endpoint. Bob will then automatically pick up the failing test context, trigger the issue resolution workflow, and post the correction result back to the Jenkins build log. This closes the automation loop without requiring any manual triage step.

Before and After: The Impact of the IBM Bob Integration

The following table captures the practical difference the IBM Bob and Tricentis integration makes across the key activities of a SAP test automation program.

Table: IBM Bob and Tricentis integration: before and after

Without IBM Bob	With IBM Bob
Test case creation is manual — a test engineer reads requirements and writes test cases one by one, typically over several days	Bob generates a complete, consistently structured test case set from requirements in under an hour, with source traceability, test data, preconditions, and step-by-step test steps
Descriptive programming module discovery requires the test engineer to manually browse the module library and cross-reference documentation	Bob searches the Tricentis module library, identifies relevant modules for the scenario, and suggests the correct call sequence with parameter values in a single conversation
Test script failures require manual triage — the tester reads the failure, opens the script, investigates the root cause, and applies the fix	Bob reads the failure, traces it to the root cause via the ToscaCloud MCP Server, applies the correction to the script, re-runs the test, and reports the outcome automatically
Regression suite maintenance after SAP patches requires days of manual review and script updates	Bob identifies scripts broken by the patch, applies corrections, and validates the corrected suite — compressing multi-day maintenance cycles to hours
CI/CD pipeline failures require a tester to pick up the failure notification, investigate, and manually re-trigger the pipeline after fixing the script	The CI/CD pipeline automatically invokes Bob when Tricentis reports a failure; Bob corrects and re-runs without human intervention, and the pipeline proceeds when the test passes
Test documentation is a separate manual effort, often neglected under time pressure	Bob generates test documentation as a by-product of every test case generation and correction activity — the documentation is always current because it is produced alongside the test work
Knowledge is concentrated in specific testers who know the module library and the SAP processes being tested	Bob acts as a shared knowledge layer — any developer or tester on the team can use natural language to query the module library, generate test cases, and apply corrections, regardless of their Tricentis expertise level

Summary and Key Takeaways

Tricentis is the platform of choice for SAP test automation at enterprise scale. IBM Bob is the AI assistant that makes that platform accessible, faster to use, and more resilient to change. The integration between them — through the ToscaCloud MCP Server and the Tricentis AI Workspace — creates a test automation capability that can keep pace with the speed of modern SAP delivery.

Three takeaways stand out from this section.

First, the IBM Bob and Tricentis integration is not a replacement for skilled test engineers — it is an amplifier. Bob handles the high-volume, repetitive, and knowledge-intensive work of test case generation, module discovery, and script correction. This frees test engineers to focus on test strategy, business process validation, and the judgement-based work that genuinely requires human expertise.

Second, the integration works best as part of a complete delivery chain. Connecting Bob to Jira or ServiceNow for issue ingestion, to Tricentis for test script management, and to Jenkins or SAP CI/CD for pipeline automation creates a fully automated quality loop. A code change enters at one end, and a validated deployment comes out the other — with IBM Bob handling the quality gate work at every step.

Third, descriptive programming in Tricentis combined with IBM Bob's module discovery capability is a significant unlock for test suite maintainability. Teams that adopt descriptive programming with Bob's assistance build test suites that are genuinely resilient to SAP change — not just automated, but sustainably automated over the full lifetime of the SAP program.

For more information, see the Tricentis section of the Additional Resources.

For related topics within this publication, see the following sections:

- IBM Bob ServiceNow Integration: For information about ServiceNow issue ingestion and the IBM Bob JWT integration pattern, see IBM Bob and ServiceNow: Integrating the AI Developer Assistant with Enterprise Change Management.
- IBM Bob Jira Integration: The *Jira* section covers Jira issue management integration with IBM Bob.

IBM Bob, SAP Signavio, and SAP LeanIX: Connecting Process Intelligence to AI-Assisted Delivery

This section shows how to connect IBM Bob to SAP Signavio and SAP LeanIX so that process models and enterprise architecture data become available directly inside an AI-assisted delivery workflow. Two integration patterns are covered — browser automation using the Playwright MCP server, and a purpose-built API-based MCP server — along with worked examples of each in use.

> **Attention:** All examples, code snippets, configuration files, demo scenarios, process models, and tool names shown in this section are for demonstration purposes only. They represent one possible implementation approach and are not standard deliverables, reference architectures, or endorsed configurations from IBM, SAP, Tricentis, or any other vendor.

Enterprise architecture and business process management have long existed in their own organizational silo. Companies invest heavily in dedicated tools — SAP Signavio for process modeling, SAP LeanIX for enterprise architecture — but developers, architects, and delivery teams rarely interact with that data during day-to-day work. The gap between what the process and architecture teams have documented and what the delivery team actually uses is one of the more persistent inefficiencies in large SAP programs.

IBM Bob's fundamental strength is context. It reads the environment it lives in and acts on it. When Bob can reach into a process repository and pull a BPMN model directly into a developer conversation — or query an enterprise architecture repository to understand which systems a new microservice must integrate with — the gap between strategy and execution narrows considerably. A developer writing an order management service no longer has to track down a process analyst to understand the approved process flow. Bob fetches it.

SAP has not released a standard MCP server for either Signavio or LeanIX. That is not a permanent blocker; it is a gap that a modest amount of Node.js fills. This section covers two integration patterns. The first uses the Playwright MCP server for browser automation — a quick start that requires no custom code but carries a meaningful per-operation cost in Bob credits. The second uses a purpose-built API-based MCP server that integrates directly with the Signavio and LeanIX REST and GraphQL APIs — more effort to set up, but dramatically more efficient and reliable at scale.

Both patterns are demonstrated using real IBM Bob conversations. By the end of the section, a developer, architect, or project manager will understand which pattern suits their team, how to implement the one they choose, and what the experience of using it looks like in practice.

Understanding the Tools

SAP Signavio Process Manager

SAP Signavio Process Manager is a cloud-based business process management platform. Organizations use it to model, document, analyze, and publish business processes. At its core, it stores process models — diagrams built in BPMN 2.0 (Business Process Model and Notation) that describe how work flows through an organization. A purchase-to-pay process, an order fulfillment flow, an HR onboarding sequence — each of these lives as a versioned model in Signavio, accessible to the whole organization through a governed collaboration environment.

Beyond the diagrams themselves, Signavio maintains a process dictionary. The dictionary holds the definitions of business terms, organizational units, activities, and IT systems that appear across multiple processes. When a process references "Credit Management" as an activity, that term links to a dictionary entry that carries its definition, owner, and related processes. This linkage between model and dictionary is what makes Signavio a governance tool rather than just a drawing application.

The platform exposes a REST API — the Signavio Process Manager API — that grants full programmatic access to models, folders, dictionary entries, and the search index. Authentication uses a JWT token obtained via a form-encoded POST to `/auth/v1/token`. Once authenticated, all subsequent calls carry the token in an `x-signavio-id` header, with the API base path following the pattern `/spm/v1/`. The token is valid for eight hours, so a well-behaved client caches it and refreshes only on expiry.

SAP LeanIX

SAP LeanIX is an enterprise architecture management platform — the system that answers questions like "which applications support this business capability?", "what is our cloud migration status?", and "which systems will be affected if we decommission this legacy platform?". It stores a graph of architectural objects — applications, interfaces, capabilities, domains, data objects, and the relationships between them — all queryable through a structured API.

The LeanIX API uses a GraphQL-based query model for complex data retrieval, combined with a REST interface for standard CRUD operations. Authentication follows an OAuth 2.0 client credentials flow. An API token is exchanged for a bearer token, which is then passed in the `Authorization` header of every subsequent call. The full API documentation, including the GraphQL schema and REST endpoints, is available through the SAP Help Portal at https://help.sap.com/docs/leanix/ea/sap-leanix-apis.

Where Signavio answers "how does this process work?", LeanIX answers "what systems make this possible, and in what state are they?". The two tools are complementary, and SAP's roadmap positions them as an integrated pair — process models in Signavio linked to capability and application data in LeanIX. For a delivery team, having IBM Bob connected to both means the developer can ask a single question and receive an answer that spans both process and architecture.

Why IBM Bob Belongs in This Picture

The value of connecting Bob to these platforms is not automation for its own sake. It is relevance. A developer working on a new capability in SAP S/4HANA should be able to ask Bob a question and get an answer grounded in the actual process and architecture documentation the organization has approved — not from a stale export that someone emailed out last quarter, but from the live repository that the process and architecture teams are actively maintaining.

That documentation already exists in Signavio and LeanIX. The missing piece is the connection. Two patterns exist for building that connection today, and they sit at opposite ends of the trade-off spectrum between speed of setup and efficiency of operation.

Integration Pattern 1 — Browser Automation via Playwright MCP

How It Works

The Playwright MCP server is a general-purpose browser automation integration available for IBM Bob. When installed, it gives Bob the ability to open a browser, navigate to URLs, take screenshots, read visible page content, click elements, and fill in forms — all under the direction of a natural language conversation.

Applied to Signavio or LeanIX, this means a user can log in to the web interface through the Playwright-controlled browser — entering their own credentials interactively, so no secrets are stored anywhere — and then ask Bob to navigate to specific processes, read their content from screenshots, or perform bulk operations across the interface. Bob interprets what it sees on screen and takes actions accordingly.

The configuration is straightforward. Adding the Playwright MCP server to Bob's `mcp.json` is all that is required:

```
{
  "mcpServers": {
    "playwright": {
      "command": "npx",
      "args": ["-y", "@playwright/mcp@latest"],
      "env": {}
    }
  }
}
```

When This Approach Makes Sense

The Playwright approach works well for one-off tasks where writing a full integration is not justified. If an architect occasionally needs Bob to summarize a specific process from Signavio, or a project manager wants to pull a process diagram description into a meeting note, browser automation is a reasonable means to that end. The interactive login means credentials never need to be configured in Bob — a meaningful security advantage for organizations with strict credential governance policies.

It also works for LeanIX and any other web-based tool that does not expose an API Bob can reach directly. If a platform lacks a usable REST API, or if access to the API requires enterprise licensing that the team does not have, the browser automation path is the practical alternative.

The Real Cost: Bob Credits

Browser automation is expensive in terms of AI inference. Every screenshot requires vision model processing. Every page interaction requires the model to interpret the rendered HTML layout. A task that a direct API call would resolve in a single round-trip can require five to fifteen model interactions when routed through a browser session — each one consuming from the user's Bob credit allocation.

This is not a theoretical concern. In practice, teams that rely on Playwright as the primary integration mechanism for frequently used tools exhaust their credits much faster than teams that invest in a proper API-based MCP server. For a task performed once a month, the cost is negligible. For a task performed multiple times a day by an active developer, the difference compounds quickly.

> **Tip:** A useful rule of thumb: if a tool has a usable REST API and a team accesses it more than occasionally, build the MCP server. Playwright is a bridge, not a foundation. Start with it to validate that the integration is useful — if the team is using it regularly after two or three weeks, that is the signal to invest in the custom server.

Integration Pattern 2 — Custom API-Based MCP Server for Signavio

The Architecture

A custom MCP server is a small Node.js process that translates Bob's tool call protocol into API calls against a target system. Bob does not call Signavio directly — it calls the MCP server, which handles authentication, constructs the appropriate HTTP request, and returns a structured result. From Bob's perspective, Signavio tools behave exactly like any other capability Bob has been given. The underlying HTTP mechanics are entirely hidden.

The architecture has a clear, deliberate separation of concerns. The MCP server owns the authentication state, handles token refresh, and abstracts all the HTTP mechanics. Bob operates purely at the intent level — "search the dictionary for entries related to credit management", or "create a new journey model in the Processes folder" — and the server handles the rest.

```
IBM Bob (Claude)

      │
      │  MCP protocol (stdio)
      ▼
signavio-mcp (Node.js process)
      │
      │  HTTPS + JWT authentication
      ▼

SAP Signavio Process Manager REST API
(api.<region>.signavio.cloud.sap/spm/v1/)
```

Building the Signavio MCP Server

The `signavio-mcp` project provides a complete, working implementation. The project is structured around two files: `signavio-client.js`, which manages authentication and HTTP requests, and `mcp-server.js`, which defines the tool schema and routes incoming requests to the appropriate API call. The project setup uses a standard Node.js ESM module:

```
"signavio-api": {
  "command": "npx",
  "args": ["-y", "signavio-mcp"],
  "env": {
    "SIGNAVIO_BASE_URL": "https://editor.signavio.com",
    "SIGNAVIO_EMAIL": "${env:SIGNAVIO_EMAIL}",
    "SIGNAVIO_PASSWORD": "${env:SIGNAVIO_PASSWORD}",
    "SIGNAVIO_TENANT": "Put Tenant ID Here"
  },
  "disabled": false,
  "metadata": {
    "description": "SAP Signavio MCP Server — models, folders, dictionary, search, export (BPMN/PNG/SVG)",
    "region": "EU",
    "workspace_url": "https://editor.signavio.com/p/hub/home"
  }
}
```

The MCP SDK (`@modelcontextprotocol/sdk`) provides the server scaffolding. `axios` handles HTTP requests. `qs` serializes the form-encoded POST body that Signavio requires for authentication. `dotenv` loads credentials from a `.env` file at startup, keeping them out of the source code entirely.

Authentication and the Client Layer

Signavio's authentication endpoint accepts a form-encoded POST body containing the email, password, tenant identifier, and a `tokenonly` flag. The response carries a JWT in the body and a `JSESSIONID` cookie. Both are needed — the JWT goes into the `x-signavio-id` header on every subsequent request, and the session cookie accompanies it. The client caches the token with an eight-hour expiry window, so repeated calls within a session do not re-authenticate unnecessarily:

```
// signavio-client.js — authentication layer (simplified)

import axios from 'axios';
import qs from 'qs';
import dotenv from 'dotenv';

dotenv.config();

export class SignavioClient {
  constructor() {
    this.baseUrl     = process.env.SIGNAVIO_BASE_URL;
    this.email       = process.env.SIGNAVIO_EMAIL;
    this.password    = process.env.SIGNAVIO_PASSWORD;
    this.tenant      = process.env.SIGNAVIO_TENANT;
    this.token       = null;
    this.sessionId   = null;
    this.tokenExpiry = null;
  }

  async authenticate() {
    // Return cached token if still valid
    if (this.token && this.tokenExpiry && Date.now() < this.tokenExpiry)
      return { token: this.token, sessionId: this.sessionId };

    const response = await axios.post(
      `${this.baseUrl}/auth/v1/token`,
      qs.stringify({ tokenonly: true, name: this.email,
        password: this.password, tenant: this.tenant }),
      { headers: { 'Content-Type': 'application/x-www-form-urlencoded' } }
    );

    // JWT from body; JSESSIONID from Set-Cookie header
    const jwtToken = typeof response.data === 'string'
      ? response.data : response.data?.token || response.data;
    const cookies  = response.headers['set-cookie'] || [];
    let sessionId  = null;
    for (const c of (Array.isArray(cookies) ? cookies : [cookies])) {
      const m = c.match(/JSESSIONID=([^;]+)/);
      if (m) { sessionId = m[1]; break; }
    }

    this.token       = jwtToken;
    this.sessionId   = sessionId;
    this.tokenExpiry = Date.now() + (8 * 60 * 60 * 1000); // 8 hours
    return { token: jwtToken, sessionId };
  }
}
```

The MCP Server and Tool Schema

The MCP server class registers every available tool and routes incoming calls to the correct client method. The SDK's `StdioServerTransport` connects Bob to the server through standard input and output — a simple, reliable channel that requires no open network port and works cleanly inside IBM Bob's process model:

```
// mcp-server.js — structure overview

import { Server } from '@modelcontextprotocol/sdk/server/index.js';
import { StdioServerTransport } from '@modelcontextprotocol/sdk/server/stdio.js';
import { CallToolRequestSchema, ListToolsRequestSchema } from '@modelcontextprotocol/sdk/types.js';
import { SignavioClient } from './signavio-client.js';

const client = new SignavioClient();

class SignavioMCPServer {
  constructor() {
    this.server = new Server(
      { name: 'signavio-api', version: '1.0.0' },
      { capabilities: { tools: {} } }
    );
    this.setupHandlers();
  }

  setupHandlers() {
    // Tool registry — lists all tools to Bob
    this.server.setRequestHandler(ListToolsRequestSchema, async () => ({
      tools: [
        { name: 'signavio_authenticate',
          description: 'Authenticate with Signavio API and get a token',
          inputSchema: { type: 'object', properties: {} } },
        { name: 'signavio_search',
          description: 'Full-text search across all Signavio content',
          inputSchema: { type: 'object',
            properties: { query: { type: 'string' },
              limit: { type: 'number', default: 20 } },
            required: ['query'] } },
        // ... 20 additional tool definitions
      ]
    }));

    // Tool dispatcher — routes calls to client methods
    this.server.setRequestHandler(CallToolRequestSchema,
      async (request) => {
        const { name, arguments: args } = request.params;
        switch (name) {
          case 'signavio_authenticate':
            return { content: [{ type: 'text',
              text: JSON.stringify(await client.authenticate()) }] };
          case 'signavio_search':
            return { content: [{ type: 'text',
              text: JSON.stringify(await client.request(
                '/search', 'GET', null,
                { q: args.query, limit: args.limit ?? 20 })) }] };
          // ... additional cases
          default:
            throw new Error(`Unknown tool: ${name}`);
        }
      }
    );
  }

  async run() {
    const transport = new StdioServerTransport();
    await this.server.connect(transport);
  }
}

new SignavioMCPServer().run();
```

Complete Tool Inventory

The server exposes twenty-two tools across four functional categories. Each tool maps directly to one or more Signavio API endpoints, and each is described to Bob with a clear, intent-oriented description so that Bob selects the right tool for the right job without needing explicit instruction from the user.

Table: Signavio MCP server — complete tool inventory across four functional categories

Tool	Category	Purpose
signavio_authenticate	Auth	Obtain and cache the JWT token for subsequent calls
signavio_search	Search	Full-text search across all Signavio content
signavio_get_root_folders	Folders	List top-level folders in the Signavio workspace
signavio_get_folder_contents	Folders	List models and sub-folders within a folder
signavio_create_folder	Folders	Create a new folder at the specified path
signavio_get_model	Models	Retrieve the full JSON representation of a model
signavio_get_model_info	Models	Retrieve model metadata (name, type, revision, author)
signavio_create_model	Models	Create a new model (BPMN, DMN, Journey, etc.) in a folder
signavio_update_model	Models	Update the JSON content of an existing model
signavio_delete_model	Models	Delete a model from the workspace
signavio_copy_model	Models	Copy a model to a different folder
signavio_move_model	Models	Move a model to a different folder
signavio_export_model_png	Models	Export a model as a PNG image
signavio_export_model_svg	Models	Export a model as an SVG vector image
signavio_export_model_bpmn	Models	Export a model as a BPMN 2.0 XML file
signavio_get_dictionary_entries	Dictionary	List all entries in the Signavio process dictionary
signavio_get_dictionary_entry	Dictionary	Retrieve a specific dictionary entry by ID
signavio_create_dictionary_entry	Dictionary	Create a new dictionary entry
signavio_update_dictionary_entry	Dictionary	Update an existing dictionary entry
signavio_search_dictionary	Dictionary	Search dictionary entries by keyword
signavio_get_model_revisions	Models	List all revision history entries for a model
signavio_get_revision	Models	Retrieve the content of a specific model revision

Credentials and Region Configuration

The server reads all configuration from environment variables. The `.env` file holds the credentials and the region-specific API endpoint. The base URL must match the geographic region where the Signavio workspace is hosted — SAP operates Signavio cloud instances in seven regions, and using the wrong regional endpoint will return authentication errors regardless of whether the credentials are correct:

Table: Signavio API regional endpoints — match base URL to the region where the workspace is hosted

Region	Base URL
Europe (EU)	https://editor.signavio.com
United States (US)	https://us.signavio.com
Australia (AU)	https://au.signavio.com
Canada (CA)	https://ca.signavio.com
Japan (JP)	https://jp.signavio.com
Singapore (SGP)	https://sgp.signavio.com
South Korea (KR)	https://kr.signavio.com

The Tenant ID (also called the Workspace ID) is found inside Signavio under **Help** → **Workspace Information**. Without the correct tenant ID, authentication may succeed but all API calls return empty results or access-denied responses. Confirming the tenant ID is the most common troubleshooting step when the server authenticates but returns no data.

> **Important:** Never commit the `.env` file to version control. Add it to `.gitignore` before the first commit. Credentials embedded in a committed file are exposed to anyone with repository access — including read-only collaborators. For team deployments, pass credentials through environment variables set at the process level or through a secrets manager, rather than from a file on disk.

Connecting the Signavio MCP Server to IBM Bob

Registering in mcp.json

Once the server is built and the `.env` file is populated, it is registered in Bob's `mcp.json` file. Two registration options exist: running from a local clone of the source code, or using npx to pull the published `signavio-mcp` package directly without maintaining a local copy:

```
// Option A — run from local source clone
{
  "mcpServers": {
    "signavio-api": {
      "command": "node",
      "args": ["/absolute/path/to/signavio-mcp/mcp-server.js"],
      "env": {
        "SIGNAVIO_BASE_URL": "${env:SIGNAVIO_BASE_URL}",
        "SIGNAVIO_EMAIL":    "${env:SIGNAVIO_EMAIL}",
        "SIGNAVIO_PASSWORD": "${env:SIGNAVIO_PASSWORD}",
        "SIGNAVIO_TENANT":   "${env:SIGNAVIO_TENANT}"
      }
    }
  }
}

// Option B — run directly via npx (no local install required)
{
  "mcpServers": {
    "signavio-api": {
      "command": "npx",
      "args": ["-y", "signavio-mcp"],
      "env": {
        "SIGNAVIO_BASE_URL": "${env:SIGNAVIO_BASE_URL}",
        "SIGNAVIO_EMAIL":    "${env:SIGNAVIO_EMAIL}",
        "SIGNAVIO_PASSWORD": "${env:SIGNAVIO_PASSWORD}",
        "SIGNAVIO_TENANT":   "${env:SIGNAVIO_TENANT}"
      }
    }
  }
}
```

After Bob restarts, the Signavio tools appear in Bob's MCP settings page under the `signavio-api` server entry. Each tool is listed with its description and an "Always allow" toggle, giving the user control over which operations Bob can perform automatically versus which ones require explicit approval on each invocation.

The Connected MCP Server in Bob Settings

The following screenshot shows the IBM Bob Settings → MCP view with the `signavio-api` server fully connected. Status is "Connected", version 1.0.0 is reported by the server, and the three tools visible in the initial list — `signavio_authenticate`, `signavio_get_root_folders`, and `signavio_get_folder_contents` — are each enabled with "Always allow" toggled on. The server is scoped to the IBM BOB workspace directory, and the `mcp.json` configuration file path is visible in the Configuration Location field.

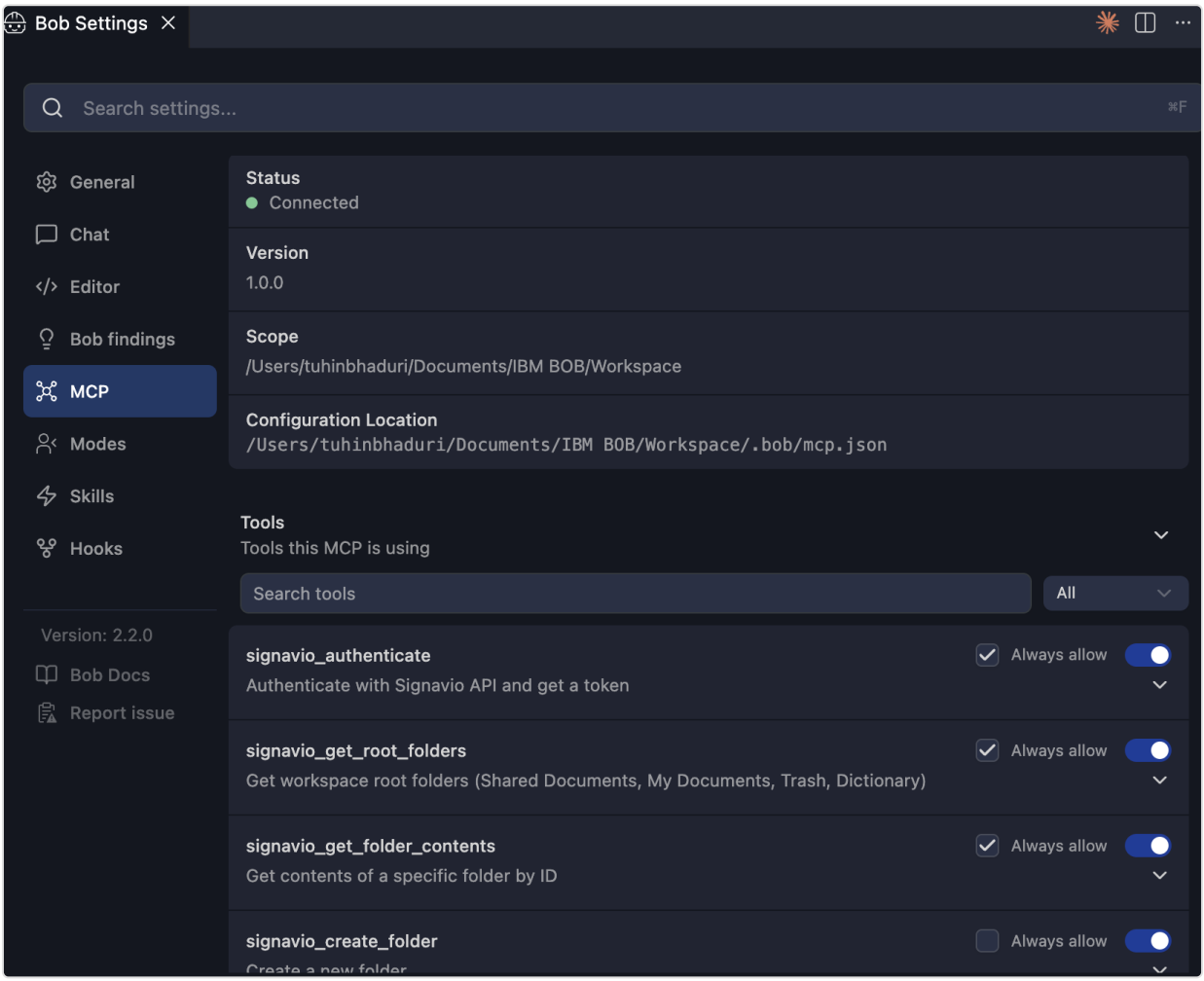


Figure: IBM Bob MCP settings with signavio-api connected

Testing the Connection

Before beginning any productive work with the Signavio integration, it is worth validating that authentication and basic API access work correctly. The `signavio-mcp` project includes a test script that validates every tool category against a live Signavio workspace:

```
# From the signavio-mcp project directory
node test-mcp-server.js
```

The script tests authentication first. A 403 response at this stage typically means wrong credentials or an incorrect tenant ID. A successful run produces a summary showing which tests passed, which were skipped due to an empty workspace, and whether any unexpected errors occurred. Running this test before registering the server in Bob is the fastest way to isolate configuration problems before they surface in a conversation.

IBM Bob in Action: Creating a Signavio Journey Model

With the `signavio-api` MCP server connected, IBM Bob can interact with Signavio through natural language instructions. The following demonstration shows Bob creating a Journey model in Signavio Process Manager — a type of process model used to represent customer or user experience flows across a set of stages and steps.

The Conversation and Tool Approval

The user opens a new Bob conversation and types a straightforward request: "Create a Journey in Signavio process manager." Bob's response is immediate and purposeful — it does not ask for clarification, but instead moves directly to action: "Let me check what's available in Signavio first."

Before proceeding, Bob presents the following tool approval prompt. This is the consent mechanism for Bob for MCP tool calls: the user can approve the specific action once, reject it, or approve all tool calls for the duration of the task. In this case, the pending action is "Signavio Authenticate (signavio-api)" — Bob needs to authenticate with the Signavio API before it can take any further steps.

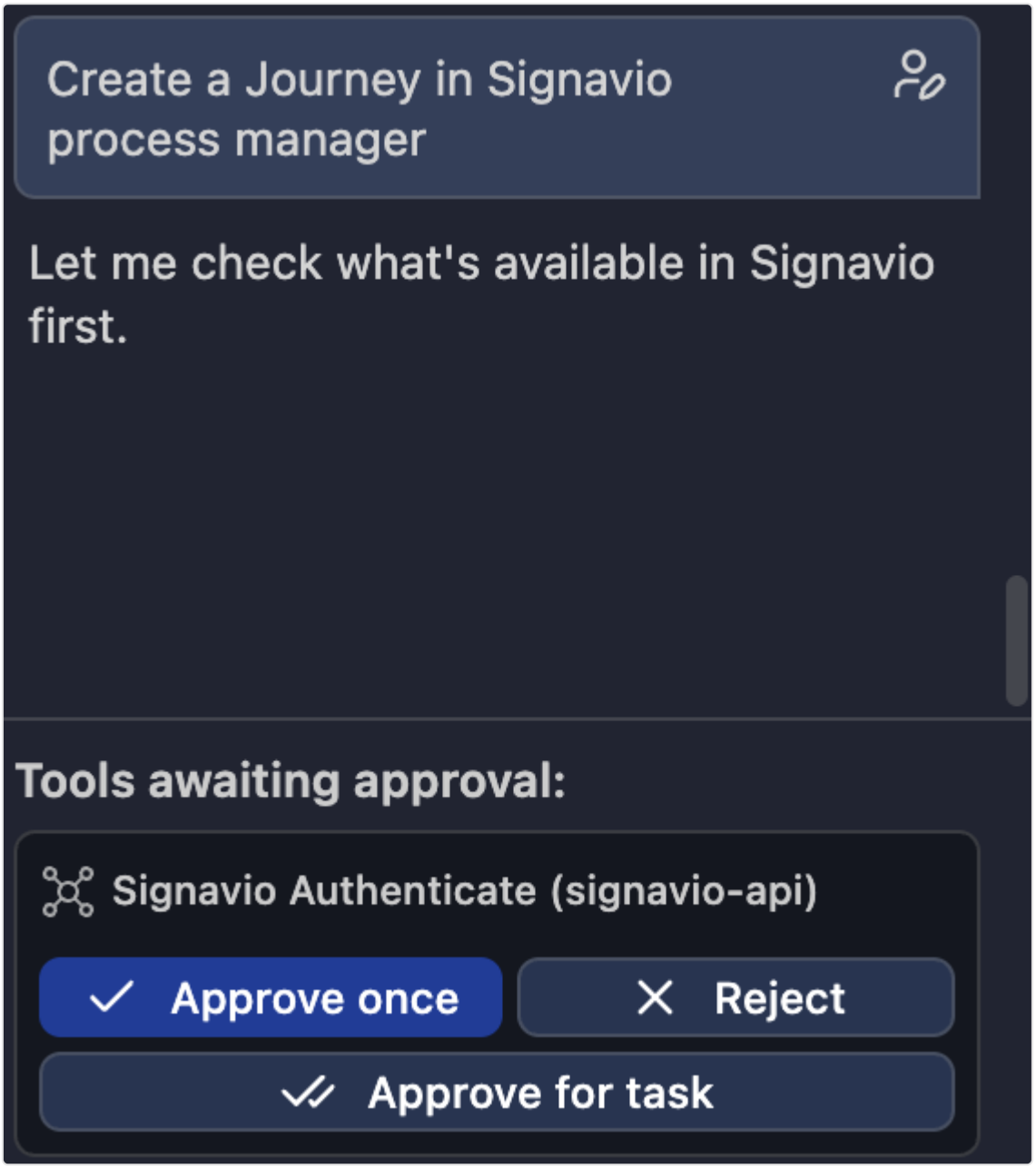


Figure: IBM Bob tool approval prompt for Signavio authentication

The three approval options have meaningfully different implications. "Approve once" allows only this single authentication call and will prompt again for subsequent Signavio tool calls. "Approve for task" grants Bob permission to use all `signavio-api` tools without further prompting for the duration of the current conversation — appropriate

when performing a multi-step Signavio workflow where re-prompting would interrupt the flow. "Reject" cancels the action and returns control to the user.

> **Note:** The "Always allow" toggles visible in the MCP Settings screenshot in the The Connected MCP Server in Bob Settings section set the default behavior for each tool globally. When "Always allow" is enabled for `signavio_authenticate`, the approval prompt shown above does not appear — Bob authenticates silently. Conversely, tools that write to Signavio (create, update, delete operations) should generally not have "Always allow" enabled without careful consideration, as this grants Bob permission to modify the Signavio workspace without explicit per-call confirmation.

The Created Journey Model in Signavio

After authentication is approved, Bob calls `signavio_get_root_folders` to understand the workspace structure, then calls `signavio_create_model` to create the Journey model in the appropriate folder. The model appears immediately in the Signavio Journey Modeler. The following screenshot shows the result — a new journey model named "Journey1" has been created and is open in the editor, ready for the user to populate with stages, steps, and goals.

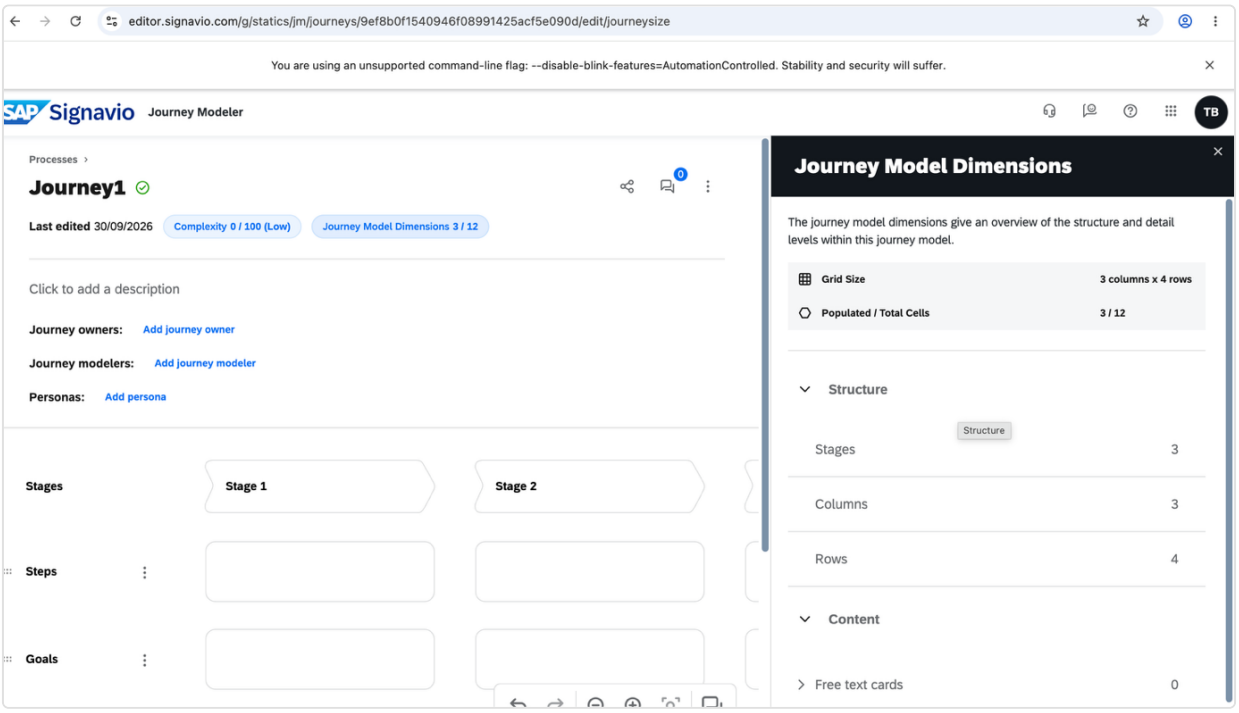


Figure: SAP Signavio Journey Modeler with Journey1 created by IBM Bob

The Journey Model Dimensions panel on the right confirms the structure of the newly created model: a 3-column by 4-row grid, with 3 of the 12 cells populated (representing the initial stage structure Bob established). The Stages row shows Stage 1 and Stage 2 are already created. The model is at Complexity 0/100 (Low) — as expected for a freshly scaffolded journey — and shows Journey Model Dimensions 3/12, reflecting the initial structure.

What took Bob approximately thirty seconds — finding the right folder, authenticating, creating the model, and structuring the initial journey scaffold — would typically require a user to navigate to the Signavio Journey Modeler, create a new model manually, name it, configure the grid dimensions, and add the initial stages. The MCP integration does not just save steps; it keeps the developer in the flow of the development conversation rather than switching to a separate browser tab and application context.

Integration Pattern for SAP LeanIX and Enterprise architecture

Enterprise architecture has always been a discipline of structured thinking in the face of relentless complexity. Organizations run hundreds — sometimes thousands — of applications, capabilities, data objects, and interfaces, all interacting in ways that no single person fully understands. The decisions made about that landscape have consequences measured in years and in millions: which systems to invest in, which to retire, which capabilities to modernize, and which technical debts to carry a little longer. Getting those decisions right requires timely, accurate, and interconnected information. Getting them wrong, or making them on the basis of stale exports and collective memory, has been one of the defining failure modes of enterprise transformation programs. SAP LeanIX is the platform purpose-built to address that challenge.

This section explains what LeanIX is, why integrating it with AI changes the nature of enterprise architecture work, how that integration enables architects to build and collaborate more effectively, and how it empowers senior leaders to make strategic decisions grounded in live architectural reality rather than in presentations assembled the night before.

What Is SAP LeanIX?

SAP LeanIX is a cloud-native Enterprise Architecture Management (EAM) platform that gives organizations a continuous, structured view of their application landscape, business capabilities, technology stack, and the relationships between them. Where traditional EA tools produced static diagrams that aged quickly, LeanIX is designed around living data — a connected graph of fact sheets that capture every meaningful element of the enterprise and the connections between them. A fact sheet is the core data object of the platform. An application, a business capability, an IT component, a data domain, an interface — each becomes a fact sheet carrying structured attributes: name, description, lifecycle status, ownership, custom fields, tags, and critically, relationships to other fact sheets. The application "SAP S/4HANA Production" links to the capability "Order Management", which links to the interface connecting to a third-party logistics provider, which in turn links to the underlying technology and the business unit that depends on it. This graph structure is what makes LeanIX indispensable for impact analysis. When a team considers decommissioning a system, LeanIX reveals its full dependency web. When an architect needs to understand which capabilities are supported by cloud-native applications versus ageing legacy systems, LeanIX surfaces that view in seconds. As part of the SAP ecosystem, LeanIX integrates natively with SAP Signavio (for business process models), SAP Cloud ALM (for project and operations visibility), and the broader SAP Business Technology Platform. It also exposes a fully documented REST and GraphQL API, enabling organizations to connect LeanIX data with the tools their teams already use — IDEs, collaboration platforms, planning systems, and now, AI assistants. "*LeanIX transforms architectural knowledge from a snapshot in a slide deck into a living, queryable graph — the single source of truth for decisions that define the enterprise*."

Why AI Integration Is Required for LeanIX

LeanIX solves the problem of capturing architectural knowledge. AI integration solves the problem of accessing that knowledge at the moment it is needed, in the context where the work is happening, without interrupting the flow of work to navigate a separate tool. Without AI integration, consuming LeanIX data requires context-switching. A developer designing an integration must pause their work, open LeanIX in a browser, navigate to the relevant capability, read through the fact sheet relationships, identify the relevant systems and their lifecycle status, and return to their development environment with the information held in short-term memory. An architect preparing a roadmap recommendation must run reports, export data, and manually correlate application portfolio information with business capability maps before they can begin the analysis. This friction is not a minor inconvenience — it is the primary reason architectural knowledge goes underused.

The data exists; the cost of retrieving it exceeds the perceived immediate benefit of retrieving it, so teams make decisions without it. AI integration collapses that retrieval cost to near zero. By connecting an AI assistant — IBM Bob used throughout this book — to the LeanIX API through a Model Context Protocol (MCP) server, every conversation the AI has can be grounded in live architectural data. A developer can ask, in plain language, which applications support a given capability. An architect can request the full dependency graph for a system under review. A program manager can query which applications are scheduled for retirement within the next eighteen months. The AI handles the translation from natural language to GraphQL query, retrieves the data from LeanIX, and returns the answer directly into the conversation — in seconds, from the tool the person was already using.

While the technical mechanism — an MCP server that wraps LeanIX's OAuth 2.0 authentication and GraphQL query interface is key here, what is most important at this stage is understanding why the integration matters: because the value of a knowledge repository is not in its storage but in its use, and AI integration makes that use effortless. How AI Integration Helps Enterprise Architects Build and Collaborate Enterprise architects operate at the intersection of business intent and technical reality. Their work requires constant cross-referencing: which capabilities does this business initiative depend on? Which applications deliver those capabilities? Which of those applications are cloud-native, and which are legacy systems carrying technical debt? Who owns them, and what is their current lifecycle status?

Answering these questions accurately and quickly is the difference between architecture that guides delivery and architecture that exists only in governance artifacts. With LeanIX connected to an AI assistant via MCP, architects gain a working partner with direct access to the architectural graph. Impact analyses that previously required hours of manual navigation — tracing dependency chains across fact sheets, correlating lifecycle data with roadmap timelines — become conversational. The architect describes what they need in natural language; the AI queries LeanIX, traverses the relevant relationships, and returns the answer with full provenance. Critically, the answer comes from the live workspace, not from a report exported last Tuesday. Collaboration also changes in character. Architecture reviews, which traditionally require architects to prepare briefing materials in advance so that non-specialists can follow the discussion, can instead be run live — with the AI retrieving relevant fact sheets, summarizing the current state of the landscape, and highlighting areas of concern as the conversation unfolds. Development teams can query architectural constraints directly from their IDEs, without waiting for an architect to be available. Architecture decisions that previously lived in someone's memory or in a shared document that no one updated become embedded in the LeanIX workspace, retrievable on demand by anyone on the team with the right context window.

The MCP server also enables architects to keep the LeanIX workspace accurate. Creating and updating fact sheets through natural language — asking the AI to register a new application, update a lifecycle status, or link a component to a capability — removes the administrative friction that causes architectural registers to drift from reality. Accuracy begets trust; trust begets use; use begets value. AI integration breaks the cycle that causes EA tooling to be actively maintained by few and passively ignored by most. Empowering Senior Leaders to Take Strategic Decisions Strategic decisions about enterprise technology — whether to consolidate platforms, retire legacy systems, invest in capability modernization, or pursue a cloud-first migration — are among the highest-stakes decisions an organization makes. They are also, routinely, made on the basis of incomplete or outdated information. The architectural data that would ground these decisions — which systems are end-of-life, which capabilities have duplicate support, which applications underpin the processes most critical to revenue — exists in LeanIX. But without a direct line from that data to the boardroom conversation, it rarely arrives in time, in the right format, or with sufficient confidence for leaders to act on it. AI integration changes that dynamic by making the architectural graph queryable in plain language by anyone with the right access — not just architects who have mastered the LeanIX interface and GraphQL query language. A CTO preparing for a board session on application rationalization can ask the AI to summarize the portfolio by lifecycle phase and identify the applications consuming the most operational cost with the lowest strategic value.

A Chief Transformation Officer can ask which capabilities have no cloud-native support and therefore represent the most significant risk to a digital transformation program. A CFO can query the estimated impact of retiring a cluster of end-of-life applications: what savings accrue, what migration effort is implied, and what dependencies require resolution first. The answers come not from an architect's PowerPoint, assembled under time pressure and inevitably reflecting someone's interpretation of the data, but from the live architectural record itself — consistent, auditable, and current. Senior leaders gain the ability to stress-test proposals, explore scenarios, and challenge assumptions in real time rather than waiting for the next briefing cycle.

The distance between the people who hold strategic intent and the people who hold architectural knowledge shrinks, and with it, the lag between insight and decision. This is, ultimately, the promise of integrating AI with enterprise architecture tooling: not that AI replaces the judgment of architects or the authority of senior leaders, but that it removes the information barriers that have historically made both groups less effective than they could be. The sections that follow show how that integration is built, how it can be extended, and what it looks like in the hands of a team doing real delivery work.

LeanIX and the Enterprise Architecture Graph

A LeanIX workspace stores architectural fact sheets — the core data objects of the platform. Each fact sheet represents one element of the architecture: an application, a capability, an interface, a business unit, a data object, or a technology. Fact sheets carry attributes (name, description, lifecycle status, owner), custom fields, tags, and relationships to other fact sheets. The application "SAP S/4HANA Production" might link to the capability "Order Management", which links to the interface that connects to a third-party logistics system.

This graph structure is what makes LeanIX useful for impact analysis. When a team needs to decommission a system, LeanIX reveals everything connected to it. When an architect needs to understand which capabilities are supported by cloud-native applications versus legacy systems, LeanIX produces that view. When IBM Bob has access to this graph, it can incorporate architectural reality into every conversation — not just the code in front of it, but the broader landscape that code fits into.

LeanIX API Foundations

LeanIX uses OAuth 2.0 client credentials for authentication. The sequence is a three-step pattern: POST to the token endpoint with the API key to receive a bearer token; pass that token in the `Authorization` header of all subsequent calls; refresh it before it expires. The token endpoint follows this pattern:

```
POST https://<workspace>.leanix.net/services/mtm/v1/oauth2/token

Body:    grant_type=client_credentials

Auth:    Basic  username=apitoken  password=<your-api-key>

Headers: Content-Type: application/x-www-form-urlencoded

Response:

{

  "access_token": "eyJhbGci...",

  "expires_in": 3600,

  "token_type": "bearer"

}
```

Once authenticated, complex data queries use the GraphQL endpoint. A query to retrieve all applications with their names and lifecycle phases looks like this:

```
POST https://<workspace>.leanix.net/services/pathfinder/v1/graphql

{
  allFactSheets(filter: {
    facetFilters: [{
      facetKey: "FactSheetTypes",
      keys: ["Application"]
    }]
  }) {
    edges {
      node {
        id
        name
        ... on Application {
          lifecycle {
            phases { phase startDate }
          }
        }
      }
    }
  }
}
```

The REST API covers CRUD operations on individual fact sheets — creating, reading, updating, and deleting architectural objects. For simple lookups, `GET /services/pathfinder/v1/factSheets/<id>` retrieves a specific fact sheet by its identifier. The full documentation, including the complete GraphQL schema and event stream subscriptions for real-time architecture change notifications, is available at https://help.sap.com/docs/leanix/ea/sap-leanix-apis.

Building a LeanIX MCP Server

A LeanIX MCP server follows the identical structural pattern as the Signavio server. The same MCP SDK, the same stdio transport, the same tool registration and dispatch mechanism — only the client layer and tool definitions change to reflect LeanIX's OAuth-based authentication and GraphQL query model. The `LeanIXClient` class handles both the OAuth exchange and request routing:

```
// leanix-client.js — illustrative outline (demo only)

import axios from 'axios';

export class LeanIXClient {
  constructor() {
    this.workspace   = process.env.LEANIX_WORKSPACE;
    this.apiKey      = process.env.LEANIX_API_KEY;
    this.token       = null;
    this.tokenExpiry = null;
  }

  get baseUrl() {
    return `https://${this.workspace}.leanix.net/services`;
  }

  async authenticate() {
    if (this.token && Date.now() < this.tokenExpiry) return this.token;
    const response = await axios.post(
      `${this.baseUrl}/mtm/v1/oauth2/token`,
      'grant_type=client_credentials',
      {
        auth: { username: 'apitoken', password: this.apiKey },
        headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
      }
    );
    this.token       = response.data.access_token;
    this.tokenExpiry = Date.now() + (response.data.expires_in * 1000);
    return this.token;
  }

  async graphql(query, variables = {}) {
    const token = await this.authenticate();
    const response = await axios.post(
      `${this.baseUrl}/pathfinder/v1/graphql`,
      { query, variables },
      { headers: { Authorization: `Bearer ${token}` } }
    );
    return response.data;
  }
}
```

The MCP server wraps this client and exposes tools such as `leanix_search_applications`, `leanix_get_fact_sheet`, `leanix_list_capabilities`, `leanix_create_fact_sheet`, and `leanix_query` — the last one passing raw GraphQL for advanced use cases. A developer asking Bob "which applications support the Order Management capability?" triggers a `leanix_search_applications` call that runs the appropriate GraphQL query and returns the results directly into the conversation.

> **Tip:** Build LeanIX MCP tools around business questions rather than API endpoints. A tool named `leanix_get_applications_for_capability` that takes a capability name and returns application names and lifecycle statuses is far more useful to Bob than a generic `leanix_graphql` tool that requires the caller to construct the query manually. The more specific and intent-oriented the tool description, the better Bob's tool selection will be.

Choosing Between the Two Integration Patterns

The decision between Playwright browser automation and a custom API MCP server is not primarily technical. It is economic and operational. Both patterns work. The question is which one the team will still be happy with after six months of regular use.

Table: Integration pattern comparison: Playwright browser automation versus custom API MCP server

Criterion	Playwright Browser Automation	Custom API MCP Server
Setup time	Minutes — install one npm package	Hours to days — build, test, deploy
Credentials in config	None — user logs in interactively	Required — stored in environment variables
Cost per operation	High — 5–15 model interactions per task	Low — single API call per tool invocation
Reliability	Fragile — breaks on UI changes	Stable — API contracts change infrequently
Read access	Full — reads anything visible in browser	Full — all REST/GraphQL endpoints
Write access	Limited — unreliable for complex forms	Full — all API write operations
Team-wide deployment	Difficult — each user must log in manually	Easy — credentials set once, shared via env
Suitable usage frequency	Occasional (weekly or less)	Regular (daily, multiple team members)

The inflection point is team usage frequency. A single architect who occasionally wants to ask Bob about a specific Signavio process model will not justify the build time for a custom server. A delivery team of ten developers who want Bob to reference process and architecture data as a normal part of their workflow absolutely will. The custom server pays for itself in recovered Bob credits within a few weeks of active daily use.

A practical approach: start with the Playwright integration to validate that the integration is genuinely useful before investing build time. Use it for the first two or three weeks. If the team is using it regularly — several times a day across multiple team members — that is the clear signal to build the proper API server. If it is being used once a fortnight by one person, the Playwright approach is entirely adequate.

A Practical Example

Consider a delivery team implementing a new credit check step in an order management process on SAP S/4HANA. The developer needs to understand the approved business process before writing any code, and the architect needs to confirm which systems are involved before designing the integration.

The Developer's View

Without the Signavio MCP server, the developer contacts the process analyst, waits for a meeting to be scheduled, receives a PDF export of the BPMN diagram by email, manually reads and interprets it, and then begins coding — having lost a day or more to the information-gathering exercise.

With the Signavio MCP server, the developer asks Bob directly in the IDE:

```
"Search Signavio for the order management process and show me the

 steps that involve credit checking."
```

Bob calls `signavio_search` with the query "order management credit", retrieves the matching model metadata, calls `signavio_get_model` to fetch the full diagram JSON, and presents the relevant process steps directly in the conversation. The developer has what they need in under a minute, without leaving the development environment and without waiting for anyone else.

The Architect's View

Before designing the credit check integration, the architect needs to understand which systems currently connect to the order management capability in the architecture, what their lifecycle status is, and whether any are scheduled for decommissioning. Without the LeanIX MCP server, this requires logging into LeanIX, navigating to the Order Management capability, reading the fact sheet relationships manually, and cross-referencing the lifecycle data.

With the LeanIX MCP server, the architect asks Bob:

```
"Which applications in LeanIX support the Order Management capability?

 Include their lifecycle status and any planned decommission dates."
```

Bob runs the appropriate GraphQL query through `leanix_get_applications_for_capability` and returns the architectural facts directly into the conversation. The architect sees the full picture — which systems are active, which are in retirement, which are the integration targets — and designs the credit check integration with accurate information rather than assumptions.

In both cases, the value is not just speed. It is accuracy and consistency. The information Bob retrieves comes from the authoritative source — the live Signavio model and the live LeanIX workspace — not from a cached export, a shared spreadsheet, or someone's memory of what was decided in a meeting three months ago.

Summary and Key Takeaways

SAP Signavio and SAP LeanIX hold organizational knowledge that developers and architects should be consulting regularly but rarely do — because reaching that knowledge requires leaving the development environment, navigating a separate tool, and translating what you find back into the context you are working in. IBM Bob, equipped with the right MCP integrations, removes that friction entirely.

The section demonstrated two paths to that integration. Playwright is the quick start: install one package, log in manually, and Bob can interact with either platform through the browser. The cost is high — each operation burns Bob credits at a rate that makes frequent use uneconomical. The API-based MCP server is the durable solution: a small Node.js process that translates Bob's intent into Signavio or LeanIX API calls, handles authentication transparently, and returns structured data at a fraction of the credit cost.

Four points merit particular emphasis as practitioners implement this section's guidance.

First, the Playwright approach and the custom MCP server are not mutually exclusive. Starting with Playwright and migrating to the custom server when usage justifies it is the practical path. The two patterns can even coexist — Playwright for tools that have no usable API, the custom server for the tools that matter most to the team.

Second, tool design matters as much as API access. MCP tools built around specific business questions — "get applications for a capability", "search processes by keyword" — are dramatically more effective in Bob's hands than generic pass-through tools that expose raw API parameters. Bob's tool selection depends on how clearly the tool description matches the user's intent.

Third, credential security cannot be an afterthought. Environment variables in `mcp.json` — using the `${env:VARIABLE_NAME}` syntax — keep secrets out of committed files. For team deployments, a proper secrets manager replaces file-based credential storage entirely. Treating credentials as secrets from the first day of the project avoids the much more difficult work of rotating exposed credentials later.

Fourth, the API contracts for both Signavio and LeanIX are stable across UI updates. A server built today against the current API surface will still be working when SAP ships a new version of the Signavio or LeanIX user interface next

quarter. Playwright integrations break whenever the UI changes. For tools that the team uses daily, this reliability difference compounds into a meaningful operational advantage over time.

The combination of IBM Bob, SAP Signavio, and SAP LeanIX gives a delivery team something that has long been aspirational: an AI assistant that understands not just the code it is helping write, but the business process that code is meant to implement and the architectural landscape it is meant to fit into. That is the connection between strategy and execution that these tools, individually, have always promised but have rarely delivered in practice.

For more information, see the IBM Documentation, SAP Documentation, and MCP (Model Context Protocol) sections of the Additional Resources.

SAP Cloud ALM MCP — Project & Change Management

Why SAP Cloud ALM MCP?

Why would an AI assistant need access to SAP Cloud ALM?

In a typical SAP implementation, SAP Cloud ALM contains valuable information about **projects, tasks, requirements, change requests, and operational health**. However, this information usually requires users to navigate through different screens and views to find what they need.

Now consider a simple request:

```
"Show me the critical tasks that are still pending for the upcoming go-live."
```

For an AI assistant such as IBM Bob to answer this question using live SAP Cloud ALM data, it needs a way to securely discover and invoke the relevant SAP Cloud ALM capabilities.

This is where **SAP Cloud ALM MCP** becomes valuable.

The MCP server acts as the bridge between IBM Bob and SAP Cloud ALM, exposing relevant capabilities as MCP tools. Bob can discover these tools, invoke the appropriate ones, retrieve the required information, and use the results to provide a meaningful response.

The interaction can be summarized as:

````
User Request
     ↓
   IBM Bob
     ↓
SAP Cloud ALM MCP
     ↓
SAP Cloud ALM
     ↓
Relevant Data / Results
     ↓
   IBM Bob
     ↓
User Response


SAP Cloud ALM MCP enables an AI assistant to move beyond simply answering questions from static knowledge

### Overview

SAP Cloud ALM (Application Lifecycle Management) is SAP's cloud-native solution for managing SAP implement

With IBM Bob connected to SAP Cloud ALM via MCP, Bob can query project status, retrieve tasks, and surface

The following list describes possible SAP delivery use cases:
- Query open implementation tasks for the current sprint
- Check the health of a SAP Cloud ALM project
- Retrieve change requests and transport details
- Understand which test cases are failing before a go-live

### Prerequisites

| Requirement | Details |
|---|---|
| IBM Bob IDE | Installed and running |
| Node.js | v18 or later |
| SAP BTP Account | With SAP Cloud ALM entitlement |
| SAP Cloud ALM API Access | Service key or OAuth client credentials |

### Creating SAP Cloud ALM API Credentials

1. Log in to your **SAP BTP Cockpit**
2. Navigate to your Cloud ALM subaccount
3. Go to **Services → Service Marketplace → SAP Cloud ALM API**
4. Create a **service instance** with plan `standard`
5. Create a **service key** — download the JSON, which contains:
   - `clientid`
   - `clientsecret`
   - `url` (token URL)
   - `endpoints` (API base URLs)


Set environment variables:

Window
```powershell
[System.Environment]::SetEnvironmentVariable('CALM_CLIENT_ID','your-clientid','User')
[System.Environment]::SetEnvironmentVariable('CALM_CLIENT_SECRET','your-secret','User')
[System.Environment]::SetEnvironmentVariable('CALM_TOKEN_URL','https://your-subdomain.authentication.eu10.
[System.Environment]::SetEnvironmentVariable('CALM_API_URL','https://eu10.alm.cloud.sap/api','User')
````

Linux/Mac
````

```
export CALM_CLIENT_ID='your-clientid'
export CALM_CLIENT_SECRET='your-secret'
export CALM_TOKEN_URL='https://your-subdomain.authentication.eu10.hana.ondemand.com/oauth/token'
export CALM_API_URL='https://eu10.alm.cloud.sap/api'
```

Configuration Levels and Opening the Config File

Configuration Levels - Where Does the Server Config Live?

Before adding the server, decide which level is right for your situation:

Level	File Location	When to Use
Global	~/.bob/mcp.json	Personal tools you want available on every project on your machine
Project	.bob/mcp.json in your project root	Team integrations tied to this engagement — commit to version control so every team member gets the same servers automatically

> **Project-level takes precedence** over global when the same server name exists in both files.

How to Open the Config File in IBM Bob (Step by Step)

1. Open IBM Bob with your project loaded
2. Click the **IBM Bob** icon in the Activity Bar to open the Bob panel
3. Click the **Settings** icon (⚙️) at the top of the Bob panel
4. Select the **MCP** tab
5. You will see two buttons near the top of the tab:
   - **Edit Global MCP**: opens `~/.bob/mcp.json` *(applies to all your workspaces)*
   - **Edit Project MCP**: opens `.bob/mcp.json` inside your current project folder *(Bob creates the file automatically if it does not exist yet)*
6. Click the appropriate button — the JSON file opens in VS Code
7. Add your server entry under the `mcpServers` object and **save the file**
8. Bob detects the change and **hot-reloads automatically** — no restart needed
9. Switch back to the **MCP** tab — your new server appears in the list within seconds

Setting Environment Variables for Credentials

All credentials must be stored as environment variables — never hardcoded in the JSON file. Reference them as `"${env:VAR_NAME}"` in `mcp.json`.

Windows (PowerShell — persistent):

```
[System.Environment]::SetEnvironmentVariable('MY_API_KEY', 'your-value-here', 'User')
```

macOS / Linux (persistent — add to shell profile):

```
echo 'export MY_API_KEY="your-value-here"' >> ~/.zshrc && source ~/.zshrc
```

> **Important:** After setting a new environment variable, restart IBM Bob so Bob picks up the new value.

Is a Cloud ALM MCP Server Available?

Check first:

1. Settings (⚙️) → MCP tab → search **"Cloud ALM"** or **"SAP"** in the marketplace
2. Check **https://mcpmarket.com** — filter by "Cloud Infrastructure" or search "SAP" (48,000+ servers)
3. Check https://smithery.ai and https://mcp.so
4. Check SAP BTP service marketplace for any SAP-published MCP tools

Likely outcome: As of 2026, an official SAP Cloud ALM MCP server may not be in the Bob marketplace. If not found, **build a custom MCP server** using the SAP Cloud ALM REST API.

Option A — Community / SAP-Published MCP Server

If available on the marketplace or npm:

```
{
  "mcpServers": {
    "cloud-alm": {
      "command": "npx",
      "args": ["-y", "mcp-server-sap-cloud-alm"],
      "env": {
        "CALM_CLIENT_ID": "${env:CALM_CLIENT_ID}",
        "CALM_CLIENT_SECRET": "${env:CALM_CLIENT_SECRET}",
        "CALM_TOKEN_URL": "${env:CALM_TOKEN_URL}",
        "CALM_API_URL": "${env:CALM_API_URL}"
      }
    }
  }
}
```

Option B — Build a Custom Cloud ALM MCP Server

When no server is available, build one against the **SAP Cloud ALM REST API**.

Key SAP Cloud ALM API Endpoints

API	What It Does
GET /imp-api/v1/projects	List all implementation projects
GET /imp-api/v1/projects/{id}/tasks	Get tasks for a project
GET /imp-api/v1/projects/{id}/tasks/{taskId}	Get task details
GET /analytics-api/v1/health	Get project health indicators
GET /changelog-api/v1/changeDocuments	List change documents / transports
GET /test-api/v1/testCases	List test cases
GET /test-api/v1/testCases/{id}/results	Get test execution results

Authentication uses **OAuth 2.0 Client Credentials** flow.

Ask Bob to Build the Server

```
"Build an MCP server that connects to the SAP Cloud ALM REST API.
 It should use OAuth 2.0 client credentials authentication.
 Expose these tools:
 1. list_projects — list all implementation projects
 2. get_project_tasks — get tasks for a given project ID
 3. get_project_health — get health indicators for a project
 4. list_change_documents — list change documents / transports
 5. get_test_results — get test case results for a project
 Use env vars: CALM_CLIENT_ID, CALM_CLIENT_SECRET, CALM_TOKEN_URL, CALM_API_URL"
```

Register in mcp.json

```
{
  "mcpServers": {
    "cloud-alm": {
      "command": "node",
      "args": ["/path/to/cloud-alm-mcp/build/index.js"],
      "env": {
        "CALM_CLIENT_ID": "${env:CALM_CLIENT_ID}",
        "CALM_CLIENT_SECRET": "${env:CALM_CLIENT_SECRET}",
        "CALM_TOKEN_URL": "${env:CALM_TOKEN_URL}",
        "CALM_API_URL": "${env:CALM_API_URL}"
      },
      "alwaysAllow": [
        "list_projects",
        "get_project_tasks",
        "get_project_health"
      ]
    }
  }
}
```

Key Tools to Implement

Tool	Description
list_projects	List all SAP Cloud ALM implementation projects
get_project_tasks	Retrieve tasks for a given project
get_task_details	Get full details of a specific task
get_project_health	Get current health/status indicators
list_change_documents	List transports / change documents
list_test_cases	List test cases for a project
get_test_results	Get test execution results

Example Prompts for SAP Projects

```
1. List all open implementation tasks for the SAP S/4HANA Finance project

2. What is the current health status of project P-001?

3. Show me all change documents in Draft status

4. How many test cases are failing in the current test cycle?

5. Which tasks are assigned to me and still in progress?

6. Summarize the delivery status of our SAP Cloud ALM project for the weekly status report
```

Troubleshooting

Problem	Cause	Fix
Token fetch fails	Wrong token URL or credentials	Re-check the service key JSON from BTP
403 on API calls	Insufficient scope	Ensure the service instance has correct scopes
Empty project list	Wrong API base URL	Verify CALM_API_URL from the service key endpoints
Rate limiting	Too many requests	Add retry logic with exponential backoff

> **Note:** The SAP Cloud ALM REST API is well-documented on SAP API Hub (https://api.sap.com) — making it an excellent candidate for a custom MCP server that the whole delivery team can share via HTTP transport.

For more information, see the IBM Documentation, SAP Documentation, and MCP (Model Context Protocol) sections of the Additional Resources.

Additional SAP MCP Server Catalog

This section focuses on additional MCP servers that are available for SAP. It provides a quick summary of the different SAP MCP servers (some provided by SAP and some by the community).

The following table is a quick reference for these MCP servers. It is not the full list; refer to SAP blogs for the latest information. Others exist, such as OSS Note check, AI solution availability check on Discovery Center, and API identification MCP servers.

Quick-Reference Summary Table

#	MCP Server	SAP Product	Delivery	License	Primary Use	Status	Supported Clients
1	BTP Administration	SAP BTP Cockpit/CLI	Remote Service	SAP Managed	Infrastructure administration, auditing, provisioning	GA	Claude Code, any MCP-HTTP client
2	CAP	SAP CAP / SAP Build Code	Open Source	Apache-2.0	AI-assisted CAP application development	GA	Cursor, Windsurf, GitHub Copilot, Cline, Claude Code
3	SAP Fiori Elements	SAP Fiori Elements / SAP Build	Open Source	Apache-2.0	AI-assisted Fiori UI generation	GA	Cursor, Windsurf, GitHub Copilot, Cline, Claude Code
4	SAPUI5	SAPUI5 / SAP Build	Open Source	Apache-2.0	AI-assisted UI5 application development	GA	Cursor, Windsurf, GitHub Copilot, Cline, Claude Code
5	MDK	SAP Mobile Services / SAP Build	Open Source	Apache-2.0	AI-assisted cross-platform mobile app development	GA	VS Code + Cline, Claude Code
6	Integration Suite MCP Gateway	SAP Integration Suite (API Management)	Managed Platform	SAP Managed	Expose any managed REST/OData API as MCP Server without code	GA	Claude Code, MCP Inspector, Joule Work Desktop
7	Principal Propagation Pattern	SAP Integration Suite + Cloud Connector + S/4HANA	Architecture Pattern	SAP Managed	Propagate end-user identity from MCP client to on-premise S/4HANA	GA	MCP Inspector, Joule Work Desktop

Server Profile Cards

The following profile cards describe each of the MCP servers and the details you need to set them up in your IBM Bob instance as required.

> **Important:** Evaluate each server against your client requirements and get approval before deploying it in client environments.

BTP Administration MCP Server

- **Delivery:** Remote SAP-hosted service (zero-footprint)
- **Description:** Production-grade remote MCP server for BTP infrastructure administration. Supports natural-language provisioning, auditing, and discovery across all BTP regions.
- **Endpoint:** `https://sso.mcp.btp.cloud.sap/mcp`
- **Client ID:** `e789ba01-5612-47ee-bfe7-79e26411c1ca`
- **Auth:** OAuth 2.0 Authorization Code + SAP IDS
- **Transport:** HTTP Streamable (MCP over HTTPS)
- **Clients:** Claude Code, any MCP HTTP client
- **Announced:** August 27, 2026

CAP MCP Server

- **Delivery:** Open Source (Apache-2.0)
- **Description:** Embeds CAP framework expertise (service definitions, CDS schemas, annotations, best practices) into AI coding assistants.
- **Part of:** SAP Build
- **Transport:** MCP stdio (IDE-local)
- **IDEs:** Cursor, Windsurf, Cline, Claude Code, GitHub Copilot

SAP Fiori Elements MCP Server

- **Delivery:** Open Source (Apache-2.0)
- **Description:** Provides Fiori Elements floor plan expertise, OData annotation vocabulary, and SAP Fiori Design Guidelines guardrails to AI coding assistants.
- **Part of:** SAP Build
- **Transport:** MCP stdio (IDE-local)
- **IDEs:** Cursor, Windsurf, Cline, Claude Code, GitHub Copilot

SAPUI5 MCP Server

- **Delivery:** Open Source (Apache-2.0)
- **Description:** Delivers UI5 control library knowledge, XML view/controller patterns, UI5 Tooling configuration, and `manifest.json` expertise to AI assistants.
- **Part of:** SAP Build
- **Transport:** MCP stdio (IDE-local)
- **IDEs:** Cursor, Windsurf, Cline, Claude Code, GitHub Copilot

MDK MCP Server

- **Delivery:** Open Source (Apache-2.0)
- **Description:** Provides MDK framework expertise (metadata-driven patterns, action chains, CLI access) for AI-assisted cross-platform mobile development.
- **Part of:** SAP Mobile Services / SAP Build
- **Transport:** MCP stdio (IDE-local)
- **IDEs:** VS Code + Cline (primary), Claude Code

Integration Suite MCP Gateway

- **Delivery:** Managed Platform (SAP Integration Suite)
- **Description:** No-code pipeline that converts any managed REST/OData API into a governed MCP server artifact deployable on Integration Cell.
- **Runtime:** Integration Cell
- **Source:** OpenAPI 3.0.x
- **Auth:** OAuth 2.0 inbound + BTP Destination outbound
- **Max tools:** 30 operations per MCP server artifact

MCP Server Selection Decision Tree

```
What is your primary goal?
  -> Manage BTP infrastructure
       -> BTP Administration MCP Server (remote, zero-setup)
  -> Accelerate SAP development with AI coding assistants
       -> Building CAP services
            -> CAP MCP Server (open-source, stdio, IDE-local)
       -> Building Fiori annotation-driven UIs
            -> Fiori Elements MCP Server
       -> Building freestyle UI5 apps
            -> SAPUI5 MCP Server
       -> Building cross-platform mobile apps
            -> MDK MCP Server
  -> Expose a business API to an AI agent at runtime
       -> On-premise S/4HANA (user identity required)
            -> Integration Suite MCP Gateway + Principal Propagation
       -> Cloud API (no user identity propagation needed)
            -> Integration Suite MCP Gateway (client credentials)
       -> S/4HANA Public Cloud
            -> Integration Suite MCP Gateway + OAuth2SAMLBearerAssertion
```

Deployment Pattern Comparison

Pattern	Servers	Transport	Hosting	Auth Model	Use Case
Remote SAP-hosted	BTP Administration	HTTP Streamable	SAP-managed cloud	OAuth 2.0 Auth Code + SAP IDS	Infra administration; zero setup
IDE-local open source	CAP, Fiori Elements, SAPUI5, MDK	stdio (local process)	Developer machine	None (local process)	AI-assisted development; offline capable
Managed gateway	Integration Suite MCP Gateway	HTTP Streamable	Integration Cell (customer BTP)	OAuth 2.0 Auth Code (inbound) + BTP Destination (outbound)	Govern enterprise API exposure to AI agents
Gateway + identity chain	Integration Suite + Cloud Connector	HTTP → mTLS (CC tunnel)	Integration Cell + on-premise CC	OAuth 2.0 → X.509 (principal propagation)	On-premise S/4HANA with full user identity propagation

Key Takeaways

- SAP has published seven MCP server implementations covering infrastructure administration, developer productivity, and enterprise API integration.
- Developer MCP servers (CAP, Fiori Elements, SAPUI5, MDK) are open-source (Apache-2.0), run locally via stdio, and work with all major AI coding assistants.
- The BTP Administration MCP Server is a zero-footprint remote service requiring only an endpoint URL and a public Client ID.
- The Integration Suite MCP Gateway converts any managed REST API to an MCP tool in six configuration steps, with no custom application code required.
- End-to-end user identity propagation to SAP S/4HANA on-premise is supported via a four-hop JWT-to-X.509 chain through SAP Cloud Connector.
- All SAP MCP servers enforce the principle of least privilege: AI agents operate under the calling user's existing IAM permissions.

Building a Custom MCP Server

Why Build a Custom MCP Server?

What happens when the MCP server you need does not already exist?

Not every enterprise integration has a ready-made MCP server. You may have an internal REST API, a custom SAP OData service, a project-specific application, or a business capability that is unique to your organization.

For example, imagine your organization has a custom SAP OData service that exposes sales order information. You want IBM Bob to answer:

```
"Find all delayed sales orders for customer 100045 and summarize the reasons."
```

If no existing MCP server provides access to this service, Bob cannot directly discover or invoke those capabilities. This is where **building a custom MCP server** becomes the right approach.

A custom MCP server acts as a controlled bridge between IBM Bob and your enterprise-specific systems. It can expose only the tools and capabilities required by your project while keeping the underlying APIs and implementation details hidden from the AI application.

A custom MCP server is particularly useful when:

- **No existing MCP server is available** for the target system.
- **Project-specific tools are required** that are not available in a standard integration.
- **Enterprise-specific business logic** needs to be exposed as AI-accessible tools.
- **A shared team integration** needs to be deployed centrally, for example using HTTP transport.
- **Internal REST APIs or SAP OData services** need to be exposed through MCP.
- **Security and access control** require a controlled interface between the AI assistant and enterprise systems.

IBM Bob can help **scaffold and implement** a custom MCP server, or you can build one manually using the development approach covered in this section.

Key Takeaways

A custom MCP server is not about rebuilding an existing integration. It is about **bridging the gap between AI and enterprise-specific capabilities when an existing MCP server does not meet your requirements**.

Overview

When no existing MCP server covers your integration need — whether it is SharePoint, SAP Cloud ALM, an internal REST API, or an SAP OData service — you build a custom MCP server. IBM Bob can scaffold and implement the server for you, or you can build it manually following the steps in this section.

A custom MCP server is the right choice when:

- No marketplace or community server exists for your target system
- You need tools scoped to your specific project or enterprise environment
- You want a shared team server deployed centrally over HTTP transport
- You need to wrap an internal or SAP OData API

Use the following guidance to decide when to build a custom MCP server:

- No marketplace or community server exists
- Tools need to be scoped to your project
- You want a shared team server over HTTP transport
- You need to wrap an internal API or SAP OData service

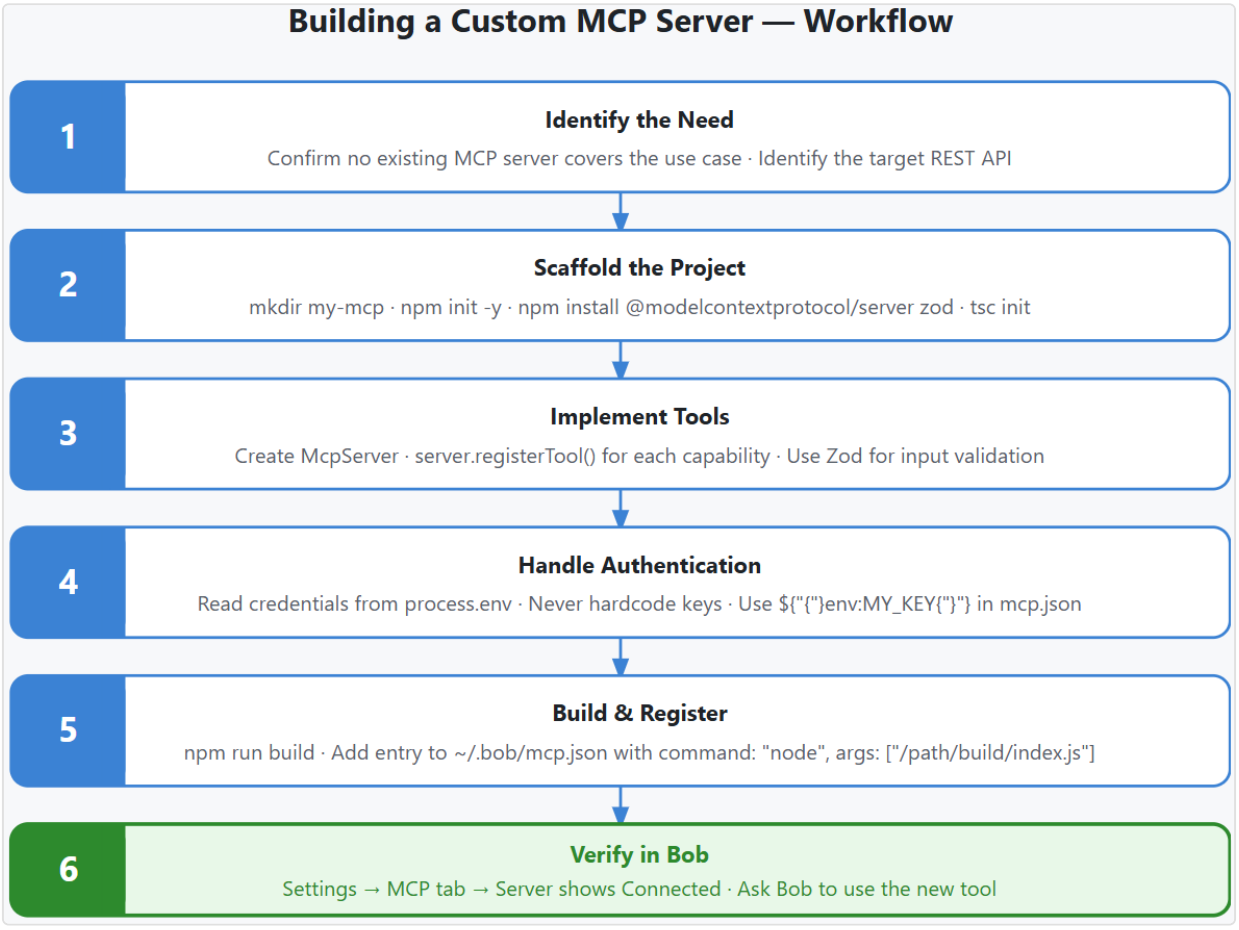


Figure: Workflow for building a custom MCP server

Prerequisites

Requirement	Details
IBM Bob IDE	Installed and running
Node.js	v18 or later
npm	Bundled with Node.js
TypeScript	Installed globally: npm install -g typescript
Target API	REST endpoint with known auth method (API key, OAuth, or similar)

Configuration Levels and Opening the Config File

Configuration Levels - Where Does the Server Config Live?

Before registering your custom server, decide which level suits your situation:

Level	File Location	When to Use
Global	~/.bob/mcp.json	General-purpose utility servers you want on every project
Project	.bob/mcp.json in your project root	Engagement-specific APIs — commit to version control alongside the server source code

> **Important:** Project-level takes precedence over global when the same server name exists in both files.

How to Open the Config File in IBM Bob (Step by Step)

1. Open VS Code with your project loaded
2. Click the **IBM Bob** icon in the Activity Bar to open the Bob panel
3. Click the **Settings** icon (⚙️) at the top of the Bob panel
4. Select the **MCP** tab
5. You will see two buttons near the top of the tab:
   - **Edit Global MCP**: opens `~/.bob/mcp.json` *(applies to all your workspaces)*
   - **Edit Project MCP**: opens `.bob/mcp.json` inside your current project folder *(Bob creates the file automatically if it does not exist yet)*
6. Click the appropriate button — the JSON file opens in VS Code
7. Add your server entry under the `mcpServers` object and **save the file**
8. Bob detects the change and **hot-reloads automatically** — no restart needed
9. Switch back to the **MCP** tab — your new server appears in the list within seconds

Setting Environment Variables for Credentials

All credentials must be stored as environment variables — never hardcoded in the JSON file. Reference them as `"${env:VAR_NAME}"` in `mcp.json`.

Windows (PowerShell — persistent):

```
[System.Environment]::SetEnvironmentVariable('MY_API_KEY', 'your-value-here', 'User')
```

macOS / Linux (persistent — add to shell profile):

```
echo 'export MY_API_KEY="your-value-here"' >> ~/.zshrc && source ~/.zshrc
```

> **Important:** After setting a new environment variable, restart VS Code so Bob picks up the new value.
>
> **Custom Server → Depends on scope:**
>
> - Use **Project** (.bob/mcp.json) for engagement-specific APIs — commit the config alongside the server source code so the team can run it
> - Use **Global** (~/.bob/mcp.json) for general-purpose utility servers you want on all workspaces

Option A — Ask Bob to Build It (Fastest) —

IBM Bob can scaffold and implement the entire server for you.

Step 1: Enable MCP Server Creation

1. **Settings** (⚙️) → **MCP** tab
2. Check **Enable MCP Server Creation**

Step 2: Ask Bob

Describe exactly what you need:

```
"Build an MCP server that connects to the Microsoft SharePoint REST API using  the Microsoft Graph API. Us
 - search_documents: search for documents by keyword across a SharePoint site
 - list_files: list files in a given document library
 - get_document_content: retrieve the text content of a file by its path
 Read credentials from env vars: MS_CLIENT_ID, MS_CLIENT_SECRET, MS_TENANT_ID,  SHAREPOINT_SITE_URL"
```

Bob will:

1. Scaffold a TypeScript project in your default MCP directory
2. Implement each tool with full API calls and error handling
3. Prompt you for credential variable names (never the values)
4. Build the project (`npm run build`)
5. Register it in your `mcp.json` automatically
6. Connect to it — tools appear in the MCP tab

Option B — Build Manually — Build Manually

Step 1: Scaffold the Project

```
mkdir my-mcp-server
cd my-mcp-server
npm init -y
npm install @modelcontextprotocol/server zod
npm install -D @types/node typescript
mkdir src
```

Update `package.json`:

```
{
  "type": "module",
  "scripts": {
    "build": "tsc && node -e \"require('fs').chmodSync('build/index.js', '755')\""
  },
  "bin": {
    "my-mcp-server": "./build/index.js"
  }
}
```

Create `tsconfig.json`:

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "Node16",
    "moduleResolution": "Node16",
    "outDir": "./build",
    "rootDir": "./src",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true
  },
  "include": ["src/**/*"]
}
```

Step 2: Implement the Server

Create `src/index.ts`:

```
#!/usr/bin/env node
import { McpServer } from "@modelcontextprotocol/server";
import { StdioServerTransport } from "@modelcontextprotocol/server/stdio";
import { z } from "zod/v4";

// Read credentials from environment — never hardcode
const API_KEY = process.env.MY_API_KEY;
const API_BASE_URL = process.env.MY_API_BASE_URL;

if (!API_KEY || !API_BASE_URL) {
  console.error("Required env vars MY_API_KEY and MY_API_BASE_URL are not set");
  process.exit(1);
}

const server = new McpServer({ name: "my-mcp-server", version: "1.0.0" });

// Register a tool
server.registerTool(
  "get_data",
  {
    description: "Retrieve data from the target system by ID",
    inputSchema: z.object({
      id: z.string().describe("The record ID to retrieve"),
    }),
  },
  async ({ id }) => {
    try {
      const response = await fetch(`${API_BASE_URL}/records/${id}`, {
        headers: { Authorization: `Bearer ${API_KEY}` },
      });
      if (!response.ok) {
        return {
          content: [{ type: "text", text: `API error: ${response.status} ${response.statusText}` }],
          isError: true,
        };
      }
      const data = await response.json();
      return {
        content: [{ type: "text", text: JSON.stringify(data, null, 2) }],
      };
    } catch (error) {
      return {
        content: [{ type: "text", text: `Failed: ${error instanceof Error ? error.message : String(error)}
        isError: true,
      };
    }
  }
);

// Register additional tools here...

async function main() {
  const transport = new StdioServerTransport();
  await server.connect(transport);
  console.error("my-mcp-server running on stdio"); // Always use console.error for logging
}

main().catch((error) => {
  console.error("Fatal error:", error);
  process.exit(1);
});
```

> **Attention:** Always use `console.error` for logging — `console.log` writes to stdout which is the MCP protocol channel. Any non-JSON stdout output will corrupt the connection.

Step 3: Handle Authentication Patterns

API Key (simplest):

```
const API_KEY = process.env.MY_API_KEY;
// Pass in headers: { 'X-API-Key': API_KEY }
```

OAuth 2.0 Client Credentials (SAP Cloud ALM, SharePoint, and similar services):

```
async function getAccessToken(): Promise<string> {
  const response = await fetch(process.env.TOKEN_URL!, {
    method: "POST",
    headers: { "Content-Type": "application/x-www-form-urlencoded" },
    body: new URLSearchParams({
      grant_type: "client_credentials",
      client_id: process.env.CLIENT_ID!,
      client_secret: process.env.CLIENT_SECRET!,
      scope: "your.api.scope",
    }),
  });
  const data = await response.json() as { access_token: string };
  return data.access_token;
}
```

Basic Auth (Jira Server and similar services):

```
const credentials = Buffer.from(
  `${process.env.USERNAME}:${process.env.API_TOKEN}`
).toString("base64");
// Pass in headers: { 'Authorization': `Basic ${credentials}` }
```

Step 4: Build

```
npm run build
```

Verify `build/index.js` exists before registering.

Step 5: Register in mcp.json

```
{
  "mcpServers": {
    "my-mcp-server": {
      "command": "node",
      "args": ["/absolute/path/to/my-mcp-server/build/index.js"],
      "env": {
        "MY_API_KEY": "${env:MY_API_KEY}",
        "MY_API_BASE_URL": "${env:MY_API_BASE_URL}"
      },
      "alwaysAllow": ["get_data"],
      "disabled": false
    }
  }
}
```

Step 6: Verify

1. Settings → MCP tab → server shows **connected**
2. Expand the server to see the registered tools
3. Ask Bob: *"Use my-mcp-server to get record with ID 123"*

Best Practices

Practice	Guidance
One tool, one job	Keep tools focused. Prefer many small tools over one large tool with a mode parameter
Use Zod for input validation	Always wrap input schemas in z.object() — this gives Bob typed, described parameters
Return isError: true on failures	Recoverable errors should set isError: true so Bob can self-correct
Never log to stdout	Use console.error exclusively — stdout is the protocol channel
Never hardcode credentials	Use process.env and ${env:VAR} in mcp.json
Use absolute paths	The args path to build/index.js must be absolute
Restart after rebuild	Click Restart (🔄) in the MCP tab after rebuilding the server

SharePoint Example — Minimal Implementation

```
server.registerTool(
  "search_sharepoint",
  {
    description: "Search for documents in SharePoint by keyword",
    inputSchema: z.object({
      query: z.string().describe("Search keyword or phrase"),
      top: z.number().optional().describe("Max results to return (default 10)"),
    }),
  },
  async ({ query, top = 10 }) => {
    const token = await getAccessToken();
    const siteId = process.env.SHAREPOINT_SITE_ID!;
    const url = `https://graph.microsoft.com/v1.0/sites/${siteId}/drive/root/search(q='${encodeURIComponen
    const response = await fetch(url, {
      headers: { Authorization: `Bearer ${token}` },
    });
    const data = await response.json() as { value: unknown[] };
    const results = (data.value || []).slice(0, top);
    return {
      content: [{ type: "text", text: JSON.stringify(results, null, 2) }],
    };
  }
);
```

Troubleshooting and Tips

Problem	Cause	Fix
Server fails to start	Build error or missing build/index.js	Run npm run build and check for TypeScript errors
"Cannot find module"	Wrong path in args	Use the full absolute path to build/index.js
Blank stdout corrupts connection	console.log used for logging	Replace all console.log with console.error
Tools not showing	Server starts but connect() not called	Ensure main() calls server.connect(transport)
Env vars missing at runtime	Variable not set before Bob starts	Set env vars, then restart Bob from that shell

Tips — Get Your Hands Dirty: Let IBM Bob Help You Troubleshoot

When you are exploring MCP for the first time—or building your first custom MCP server—**do not expect everything to work perfectly on the first attempt**.

You will likely encounter errors that you have never seen before: configuration issues, authentication failures, incorrect parameters, transport errors, server startup failures, JSON configuration problems, API connectivity issues, or unexpected tool responses. This is a normal part of the learning process.

One of the most useful capabilities of **IBM Bob** is that you do not have to solve these problems alone.

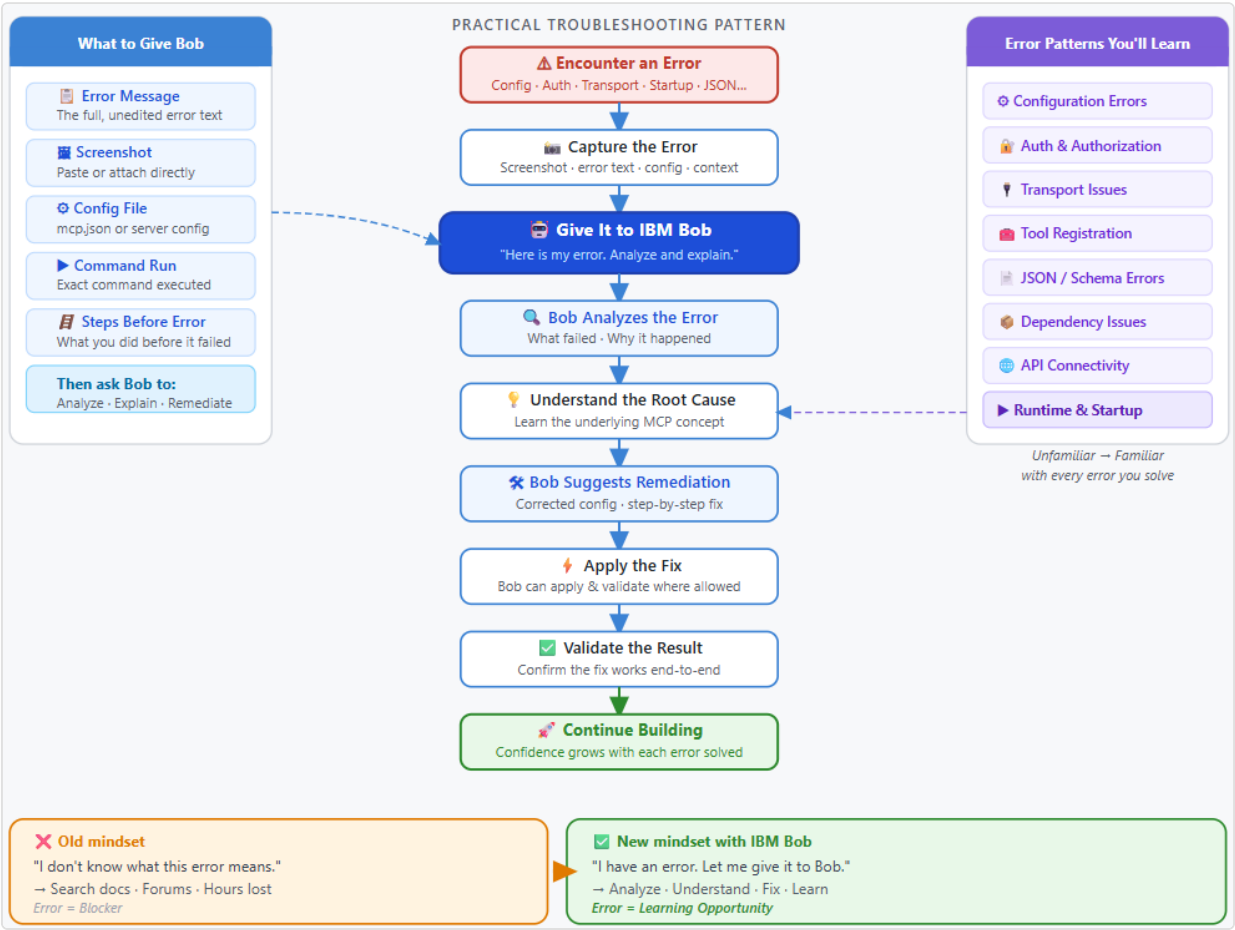


Figure: Troubleshooting MCP errors with IBM Bob

Don't Just Read the Error — Give It to Bob

When you encounter an error, resist the temptation to immediately start searching through multiple documentation pages or forums.

Instead, give the error directly to IBM Bob.

You can simply provide:

- The **complete error message**
- A **screenshot of the error**
- The relevant **configuration file**
- The command you executed
- The steps you performed immediately before the error occurred

For example:

```
"I am trying to start my MCP server, but I am getting the following error."
```

Then paste the error or attach a screenshot.

IBM Bob can analyze the error, explain **what went wrong, why it happened, and what you should do next**.

More importantly, the troubleshooting process can become interactive. You can ask Bob to inspect the configuration, suggest the required changes, explain the impact of those changes, and guide you through the remediation step by step. Where the environment and permissions allow it, Bob can also help **apply the required changes and validate the result**.

A Practical Troubleshooting Pattern

When working hands-on with MCP, follow this simple pattern:

```
Encounter an Error
        ↓
Capture the Error / Screenshot
        ↓
Give it to IBM Bob
        ↓
Bob Analyzes the Error
        ↓
Understand the Root Cause
        ↓
Bob Suggests Remediation
        ↓
Apply the Fix
        ↓
Validate the Result
        ↓
Continue Building
```

This changes the way you learn.

Instead of thinking:

```
"I don't know what this error means."
```

Think:

```
"I have an error. Let me give it to Bob and understand what is happening."
```

Turn Errors Into Learning Opportunities

This is particularly valuable when you are building a custom MCP server because many errors are actually **learning opportunities**.

For example, suppose your MCP server fails to start because of an invalid configuration.

Rather than simply asking:

```
 "How do I fix this?"
```

Ask IBM Bob:

```
 "Analyze this error. Explain the root cause, identify the incorrect configuration, suggest the corrected
```

Now you are not just fixing the problem—you are **learning the underlying MCP concept**.

Over time, you will start recognizing patterns:

- Configuration errors

- Authentication and authorization failures
- Transport-related issues
- Tool registration problems
- Incorrect JSON or schema definitions
- Dependency issues
- API connectivity problems
- Runtime and startup failures

The errors that initially look unfamiliar gradually become familiar patterns.

Key Takeaway

When getting your hands dirty with MCP, **don't be afraid of errors**. An error is simply feedback from your implementation.

Give IBM Bob the **error message, screenshot, configuration, or relevant context**, and let it help you analyze the problem, understand the root cause, identify the remediation, and—where supported—apply and validate the fix.

> **Note:** The goal is not to avoid errors while learning MCP. The goal is to use every error as an opportunity to understand MCP better and build with confidence.

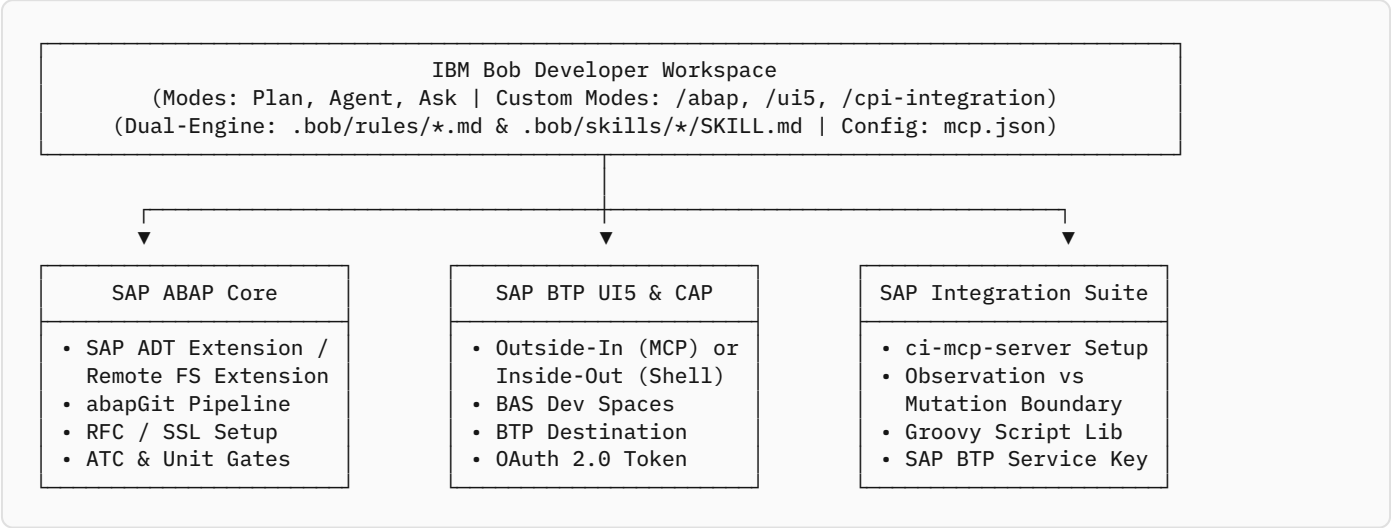
Hands-on environment setup for governed SAP development with IBM Bob

Establishing an AI-assisted development environment across an enterprise SAP landscape requires more than just installing software; it demands a disciplined, reproducible, and secure operational framework. Whether generating ABAP artifacts on an S/4HANA core, developing cloud-native SAPUI5/CAP applications in SAP Business Application Studio (BAS), or building integration flows in SAP Integration Suite (Cloud Integration / CPI), practitioners need clear, step-by-step guidance on how to bridge IBM Bob with live systems while strictly adhering to enterprise guardrails.

This section provides the complete, practical hands-on setup guide for three primary SAP development disciplines:

1. **SAP ABAP Development** (On-Premises / RISE / ABAP Cloud)
2. **SAP BTP UI5 and CAP Development** (Node.js / Cloud Application Programming Model)
3. **SAP Integration Suite / CPI** (iFlows and Groovy Scripting)

With the concrete configuration files, permission policies, MCP registration schemas, and verification checklists in this hands-on section, development teams can achieve rapid productivity gains while preserving standard change control, zero-trust security, and auditability.

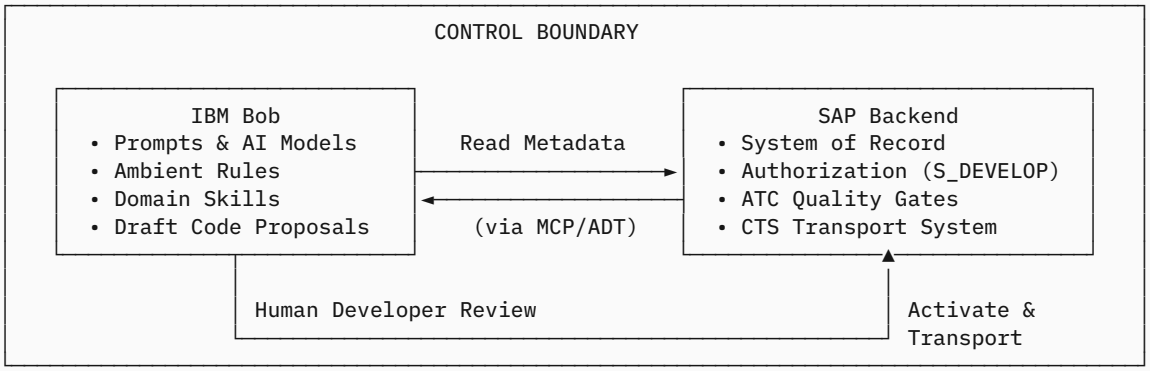


What this section covers

- **Prerequisites and Boundary Architecture**: Landscape requirements, credential isolation principles, and the four-party control model (IBM Bob, IDE Extensions, MCP Middleware, and SAP Backend).
- **SAP ABAP Hands-on Setup**:
  - Choosing between the SAP ADT Extension and ABAP Remote File System.
  - Configuring non-production RFC/HTTP destinations and workspace package boundaries.
  - Setting up read-only ADT MCP context servers.
  - Implementing ambient Clean Core rules and the `abap-coding-guidelines` skill.
- **SAP BTP UI5 and CAP (Node.js) Hands-on Setup**:
  - Configuring Category 1 (Outside-In): Deploying the local `bas-mcp-server` bridging Bob to SAP Business Application Studio via OAuth 2.0 / BTP destinations.
  - Configuring Category 2 (Inside-Out): Installing and authenticating the IBM Bob CLI Shell natively within a BAS Dev Space terminal.
- **SAP Integration Suite (CPI) Hands-on Setup**:
  - Provisioning SAP BTP Process Integration runtime instances and service keys.
  - Registering and configuring `ci-mcp-server` with explicit separation between read-only observation tools and human-gated mutation actions.
  - Setting up the dedicated CPI developer mode.
- **Verification, Diagnostics and Troubleshooting**: Smoke test suites, ATC quality gates, and error remediation matrices for network, token, and authorization failures.

Architecture, prerequisites, and control boundaries

Before initiating any local or cloud setup, the control boundaries between IBM Bob and SAP systems must be explicitly established.



The four-party operational model

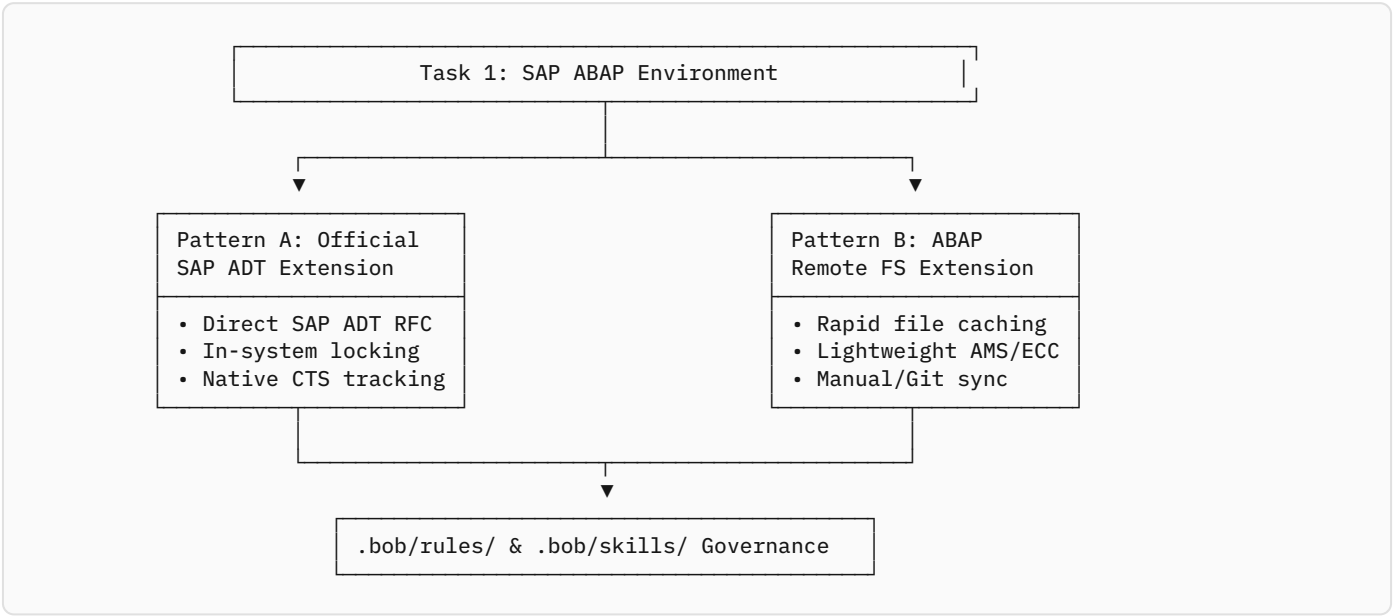
1. **IBM Bob Client**: Accepts developer instructions, evaluates workspace context, applies active Rules and Skills, and generates draft code or specifications.
2. **Connectivity Extensions / CLI**: Facilitates workspace-to-system communication (SAP ADT, Remote FS, BAS Shell, or Git).
3. **Model Context Protocol (MCP) Middleware**: Provides controlled, structured, and read-oriented access to metadata, DDIC structures, and tenant logs.
4. **SAP Backend (S/4HANA / BTP / CPI)**: Remains the sole authority for syntax checking, authorization validation, object activation, transport release, and runtime execution.

Master prerequisites checklist

Component	Minimum Requirement	Notes
IBM Bob	Version 2.0.5 or higher	Configured with enterprise sign-in and approved modes.
SAP ABAP System	SAP NetWeaver 7.31+ or SAP S/4HANA 1709+	Non-production development client; assigned developer key.
SAP BTP Account	Cloud Foundry Subaccount	Entitlements for Business Application Studio (BAS) and Integration Suite.
Local Runtimes	Node.js LTS (v18.x or v20.x), Git CLI	Required for running local MCP connectors and Git synchronization.
Security and Auth	Individual named credentials	Strict Rule: Never use production credentials, shared service users, or hardcoded secrets in configuration files.

Task 1: Hands-on setup for SAP ABAP development

This task establishes the development environment for classical ABAP, ABAP Cloud, and S/4HANA Clean Core development.



Extension installation and destination configuration

Step 1: Installing the approved ABAP extension

1. Open IBM Bob and navigate to the **Extensions View** (`Ctrl+Shift+X` on Windows/Linux, `Cmd+Shift+X` on macOS).
2. Search for the officially approved extension identifier (e.g., `SAP.adt-decomp` or community-approved `abap-remote-fs`).
3. Verify publisher authenticity and install the extension.

Step 2: Configuring the SAP development destination

Create or update the connection configuration in your workspace settings (`.vscode/settings.json` or `.bob/settings.json`):

```
{
  "abap.remote.fs": {
    "connections": {
      "S4H_DEV": {
        "url": "https://s4h-dev.internal.corp:44300",
        "client": "100",
        "user": "DEV_USER",
        "allowSelfSigned": false,
        "custom": {
          "sap-client": "100",
          "sap-language": "EN"
        }
      }
    }
  }
}
```

> **Note:** Credentials should be supplied via secure OS credential storage prompts during connection, never hardcoded in settings.

Configuring the workspace and package scoping

1. In the extension explorer, connect to `S4H_DEV`.
2. Add the designated customer package root (e.g., `ZCORE_SALES` or `$TMP` for initial disposable tests).
3. Assign your assigned Workbench Transport Request (e.g., `FS3K900123`) before creating or modifying any objects.

Implementing ABAP rules and skills

Configuring ambient rule: `.bob/rules/abap-clean-core.md`

Create the persistent rule file in your project workspace:

```
---
description: SAP Clean Core and ABAP Cloud Standards
alwaysApply: true
---

# SAP ABAP Coding & Clean Core Standards

## Language Scope & Syntax
- Target ABAP Cloud / S/4HANA Modern ABAP.
- Use inline data declarations `DATA(...)`, table expressions, and string templates `|...|`.
- Prohibit obsolete statements: `MOVE`, `COMPUTE`, `CHECK` (outside loops), `TABLES`, and header lines.
- Always escape ABAP SQL host variables with `@` (e.g., `WHERE vbeln = @lv_vbeln`).

## Clean Core Mandates
- Do NOT reference unreleased standard database tables directly (e.g., `MARA`, `VBAK`, `KNA1`).
- Consume data exclusively via SAP Released CDS Interface Views (e.g., `I_Product`, `I_SalesOrder`).
- Ensure all custom development is created within customer namespaces (`Z*` or `Y*`).

## Security & Database Performance
- Guard all `FOR ALL ENTRIES` queries with an explicit `IF lt_keys IS NOT INITIAL` check.
- Never use `SELECT *`; explicitly enumerate required fields.
- Include mandatory `AUTHORITY-CHECK` statements or DCL annotations for CDS views.
```

Configuring domain skill: `.bob/skills/abap-guidelines/SKILL.md`

Create the specialized skill for scaffolded ABAP development:

```
---
name: abap-guidelines
description: Expert skill for generating Clean Core ABAP, CDS View Entities, and RAP Business Objects
---

# ABAP Engineering Playbook

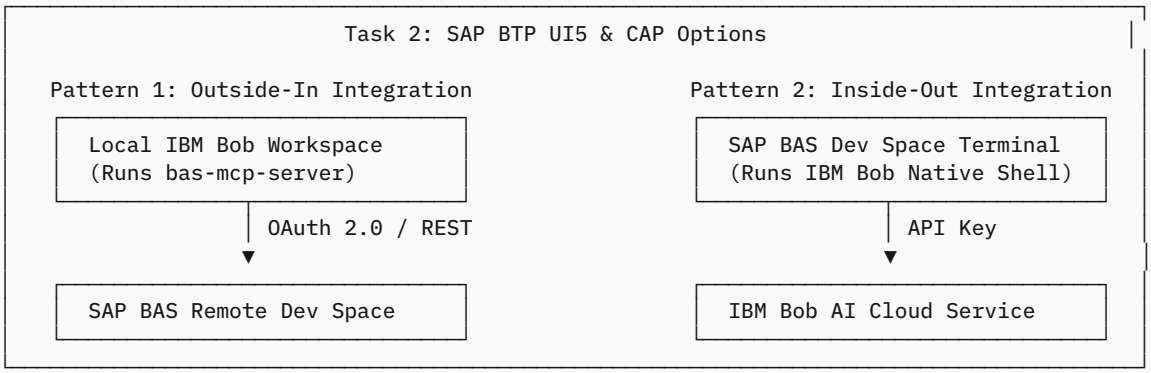
## Input Validation
- Confirm target SAP release (S/4HANA 2022+ / ABAP Cloud).
- Validate package assignment, transport ID, and interface CDS views.

## Generation Sequence for RAP Objects
1. CDS Data Model (`ZI_SalesOrderTP`) with semantic annotations.
2. Behavior Definition (`BDEF`) with lock master and authorization definitions.
3. Behavior Implementation Class (`BP_I_SALESORDERTP`) using Entity Manipulation Language (EML).
4. Service Definition (`SRVD`) and Service Binding (`SRVB`).

## Quality Verification Checklist
- [ ] Syntax check clean with zero warnings.
- [ ] ATC Check variant `ABAP_CLOUD_READINESS` passes with no Priority 1/2 errors.
- [ ] ABAP Unit test class implemented with positive and negative assertions.
```

Task 2: Hands-on setup for SAP BTP UI5 and CAP development

SAP BTP development in SAP Business Application Studio (BAS) can be configured using one of two architectural patterns:



Pattern 1

Outside-In Integration (Local Bob → BAS MCP Server)

In this pattern, IBM Bob runs on the local workstation and orchestrates SAP Business Application Studio through a purpose-built MCP server.

Step 1: Register `bas-mcp-server` in `mcp.json`

Configure the MCP server in your workspace configuration (`.bob/mcp.json`):

```
{
  "mcpServers": {
    "bas-mcp-server": {
      "command": "node",
      "args": ["/workspace/bas-mcp-server/dist/index.js"],
      "env": {
        "BAS_API_ENDPOINT": "https://<subaccount>.eu10.applicationstudio.cloud.sap/api/v1",
        "BTP_AUTH_URL": "https://<subaccount>.authentication.eu10.hana.ondemand.com/oauth/token",
        "BTP_CLIENT_ID": "sb-bas-client-id",
        "BTP_CLIENT_SECRET": "${env:BTP_BAS_CLIENT_SECRET}",
        "BAS_DEV_SPACE_ID": "ws-xv84k"
      }
    }
  }
}
```

Step 2: Validate Exposed BAS MCP Tools

Ensure Bob discovers the registered BAS tool suite:

- `bas_list_files`: Browse remote project structure within the Dev Space.
- `bas_read_file`: Inspect CDS models (`schema.cds`) and UI5 controller logic.
- `bas_write_file`: Deploy generated UI5 views or CAP handlers directly to BAS.
- `bas_execute_command`: Run `cds build`, `npm test`, or `mbt build` remotely.

Pattern 2

Inside-Out Integration (IBM Bob Shell in BAS Terminal)

In this lightweight pattern, the IBM Bob CLI is installed directly within the cloud-hosted BAS Dev Space container.

Step 1: Generate an IBM Bob API Key

1. In your enterprise IBM Bob portal, navigate to **User Settings > API Keys**.
2. Click **Generate New Key**, assign a label (e.g., `SAP-BAS-DevSpace`), and copy the token.

Step 2: Install and Link the Bob Shell in the BAS Terminal

Open the BAS Dev Space terminal and execute:

```
# 1. Install IBM Bob CLI globally inside the dev space
npm install -g @ibm/bob-cli
OR
curl -fsSL https://bob.ibm.com/download/bobshell.sh | bash

# 2. Configure the API key authentication
export BOB_API_KEY="your-generated-token"
echo "export BOB_API_KEY=\"your-generated-token\"" >> ~/.bashrc

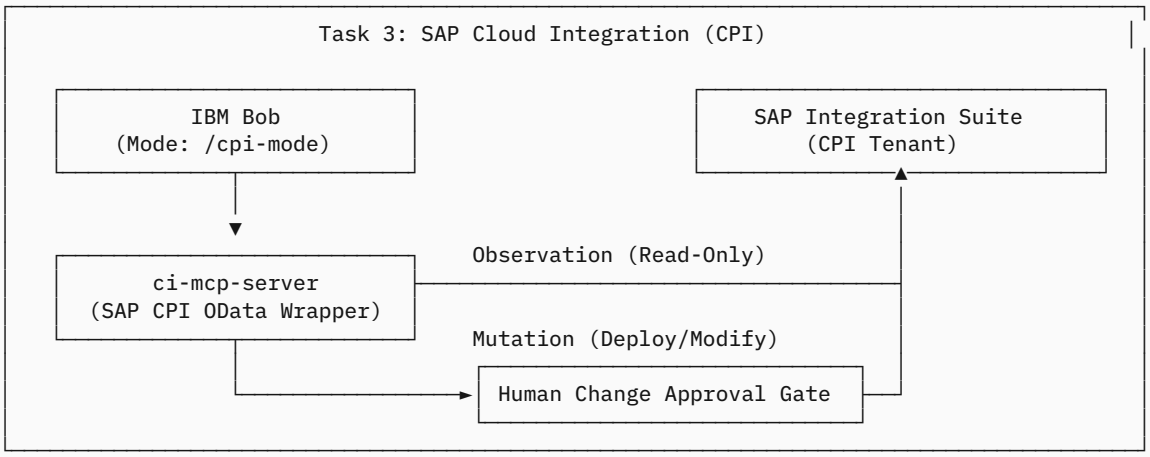
# 3. Verify connection to the IBM Bob service
bob --version
bob whoami
```

Step 3: Run AI-Assisted Commands Inside the Dev Space

```
# Generate a new CAP service handler with literate instructions
bob "Create a CAP event handler for CatalogService in srv/cat-service.js validating order total against cr
```

Task 3: Hands-on setup for SAP Integration Suite (CPI)

This Task configures governed AI assistance for SAP Cloud Integration (CPI) flows and Groovy scripting.



Provisioning SAP BTP service key for CPI

1. In the SAP BTP Cockpit, navigate to your Integration Suite subaccount.
2. Go to **Services > Instances and Subscriptions**.
3. Create an instance of **Process Integration Runtime** with plan `api`.

4. Create a Service Key with roles: `WorkspacePackagesEdit`, `MonitoringDataRead`, `AuthGroup_Administrator`.
5. Download the JSON credential payload containing `clientid`, `clientsecret`, `tokenurl`, and `baseurl`.

Registering `ci-mcp-server`

Add the configuration to `.bob/mcp.json`:

```
{
  "mcpServers": {
    "ci-mcp-server": {
      "command": "npm",
      "args": ["start"],
      "cwd": "/workspace/ci-mcp-server",
      "env": {
        "CPI_HOST": "https://<tenant>-tmn.integrationsuite.cfapps.eu10.hana.ondemand.com",
        "CPI_TOKEN_URL": "https://<subaccount>.authentication.eu10.hana.ondemand.com/oauth/token",
        "CPI_CLIENT_ID": "sb-cpi-api!t123",
        "CPI_CLIENT_SECRET": "${env:CPI_SERVICE_SECRET}"
      }
    }
  }
}
```

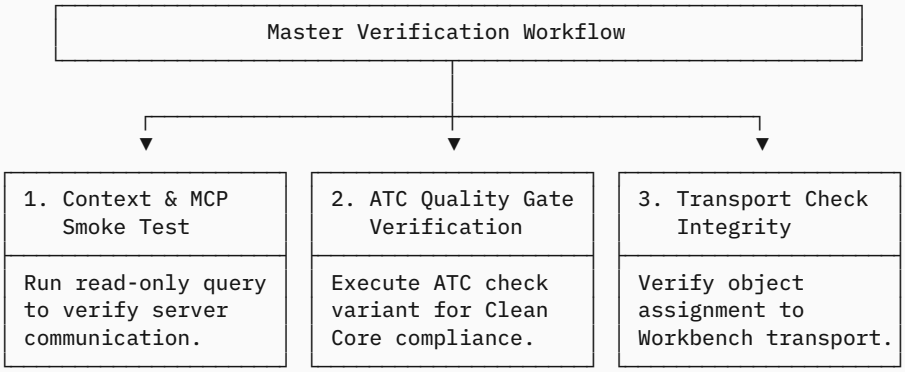
CPI governed mode definition

Add the specialized CPI mode to `.bob/custom_modes.yaml`:

```
- id: cpi-mode
  name: "SAP CPI Integration Engineer"
  roleDefinition: "You are an expert SAP Cloud Integration architect. You design iFlows, generate robust G
  groups:
    - read
    - edit
    - ["mcp", { "server": "ci-mcp-server" }]
    - skill
    - todo
  customInstructions: |
    - Always separate observation (reading iFlow metadata, message logs) from mutation.
    - Never attempt automated deployment to production without an explicit human approval gate.
    - All Groovy scripts must implement proper null-safe XML/JSON parsing, SLF4J logging, and exception bu
```

Verification, quality gates, and troubleshooting

To ensure the environment is correctly configured and compliant with enterprise policies, execute the following verification procedures.



Verification checklist

Phase	Test Action	Expected Result	Pass/Fail
Connectivity	Execute bob whoami or test extension ping.	Authenticated successfully with valid identity.	[ ]
MCP Inspection	Call read-only MCP tool on test package/iFlow.	Structured metadata returned with zero credential leaks.	[ ]
Rule Enforcement	Prompt Bob to generate a query with SELECT * FROM MARA.	Bob refuses and refactors to use released view I_Product.	[ ]
Skill Execution	Request a new RAP Business Object skeleton.	Generates CDS, BDEF, and Behavior Class in sequence.	[ ]
Transport Gate	Verify object assignment in SAP GUI / ADT.	Object is locked under the designated Workbench transport.	[ ]

Common troubleshooting scenarios

Issue Symptom	Probable Root Cause	Recommended Remediation
RFC_ERROR_LOGON_FAILURE (Task 1)	Expired developer password or incorrect SAP client ID.	Verify credentials via SAP GUI; confirm client parameter in settings.json.
401 Unauthorized / Invalid OAuth Token (Task 2 and 3)	BTP Service Key expired or invalid BTP_CLIENT_SECRET.	Regenerate the service key in BTP Cockpit; update local environment variables.
Tool call blocked: mutation not permitted	Mode configuration enforces observation-only mode.	Request explicit human approval before triggering deployment/mutation tools.
ATC Check Failure: Clean Core Violation	Generated code accessed non-released base tables.	Re-prompt Bob with @/rules/abap-clean-core.md to swap table for standard CDS view.

Adoption roadmap: Crawl, Walk, Run

Most enterprises do not fail at AI adoption because the technology is wrong. They fail because they try to do too much at once, or they treat adoption as a tool deployment rather than a cultural shift. IBM Bob is not a chatbot you install and declare complete. It is an engineering platform that deepens in value the more deliberately it is embedded into your development culture, your governance model, and your team's day-to-day habits.

This section introduces the Crawl, Walk, Run framework as the recommended sequence for adopting IBM Bob in an SAP enterprise context. Each phase has a clear definition, a set of measurable outcomes, and a natural transition point that tells you when you are ready to move forward. The framework applies equally to greenfield SAP implementations and to organizations with decades of existing ABAP, BTP, and integration landscape behind them.

The audience for this section spans the full project team. Developers will find the phase-by-phase capability map useful. Architects will find the environment and governance decisions in the Walk phase directly actionable. Project managers will see where dependencies lie and how to sequence workstreams. Senior leaders will understand why the investment in the early phases — which can feel slow — is what makes the Run phase genuinely transformative.

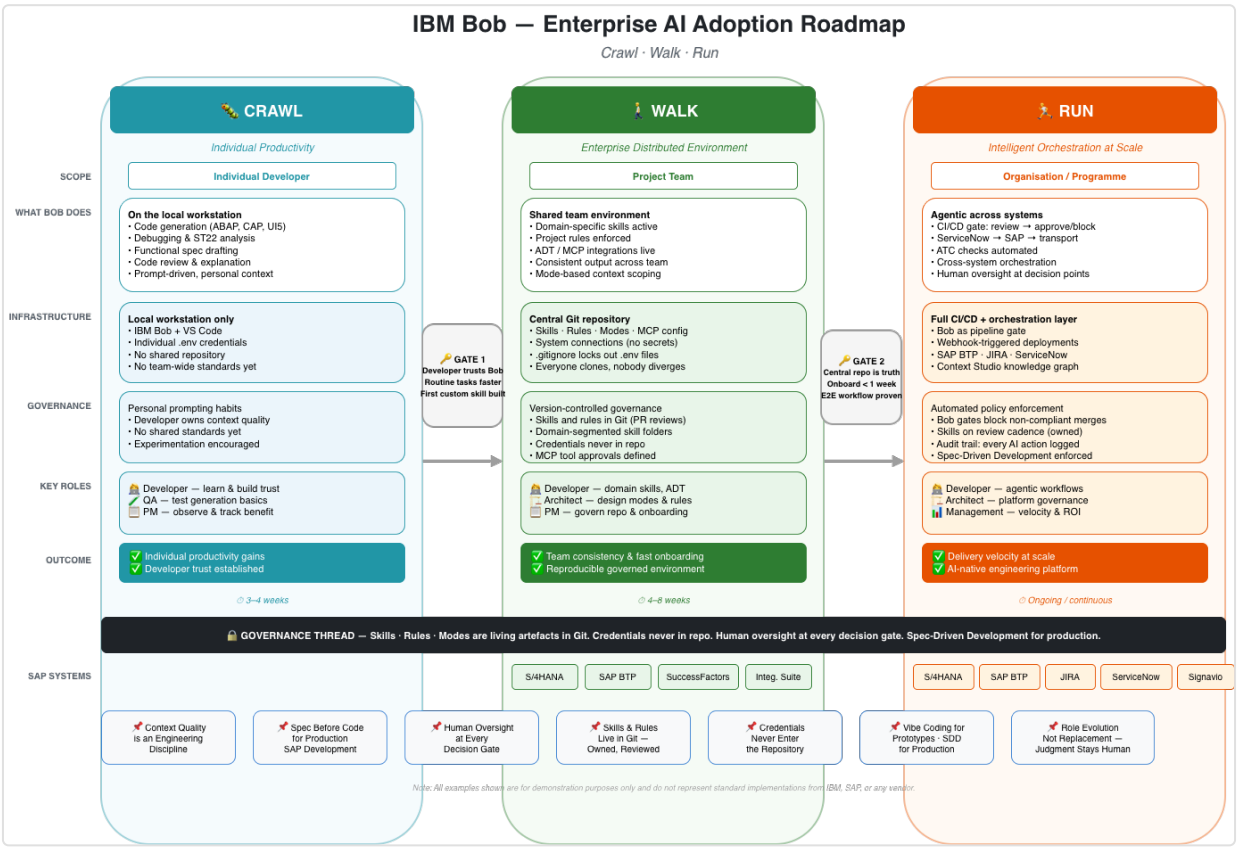
> **Attention:** All examples and demonstrations in this section are illustrative. They do not represent standard IBM, SAP, or vendor reference implementations.

Why a phased approach matters

Ask any team that has tried to deploy an AI coding assistant enterprise-wide in a single sprint how it went. The answers cluster around the same problems: inconsistent output quality, developer distrust, governance gaps, and a gradual drift back to old ways of working. The technology was fine. The sequencing was not.

IBM Bob is capable of operating across the full software development lifecycle — from requirements analysis through deployment and production monitoring. That breadth is a strength in the steady state. Early in adoption, it is a risk. Too many degrees of freedom, too little shared context, and every developer inventing their own prompting strategy produces noise rather than productivity.

The Crawl, Walk, Run framework constrains that problem. It makes deliberate choices about what to expose, what to standardize, and what to defer — so that each phase builds confidence and context that the next phase depends on.



IBM Bob — Enterprise AI adoption roadmap: Crawl, Walk, Run

Phase 1 — Crawl: Individual productivity

What the Crawl phase is

The Crawl phase is about one thing: getting each member of the team productive with IBM Bob in their own work, without requiring any shared infrastructure, governance tooling, or cross-team coordination. It is individual. It is exploratory. And it is intentionally bounded.

A developer in the Crawl phase uses Bob on their local workstation. They learn how to write specific, well-scoped prompts. They discover which capabilities — code generation, code review, documentation, debugging — are immediately useful for their SAP domain. They build intuition about when Bob's first response is good enough and when it needs one more iteration of context.

The Crawl phase is complete when an individual can answer yes to three questions: Do I know what Bob can do? Do I know how to ask for it clearly? Am I personally faster and more accurate on routine tasks because of it?

What developers learn in Crawl

The capabilities a developer needs to internalize in this phase are not complex, but they require practice. Prompt specificity matters enormously. A request like "fix the billing issue" produces a very different quality of response than "in function module `Z_INV_CREATE`, the customer invoice is not being generated when the payment terms field is blank; diagnose the cause and propose a fix consistent with SAP standard SD processing." The second request gives Bob the context it needs to reason accurately.

Beyond prompting, developers learn Bob's mode system, how skills provide domain-specific guidance, and how the agent reasons about code it has been shown versus code it has only been told about. This matters in SAP environments, where a function module might span hundreds of lines of legacy ABAP with implicit dependencies that are only visible when the actual code is in context.

In practice, the Crawl phase tasks are concrete and immediately valuable:

- Debugging a failed invoice generation or procurement exception
- Generating boilerplate ABAP for a new enhancement spot
- Writing a functional specification from a JIRA ticket
- Reviewing a colleague's code changes before a transport
- Explaining what a dense legacy ABAP program does, in plain language

None of these require shared infrastructure. All of them produce immediate, measurable value for the individual.

Productivity at the individual level

The outcome of the Crawl phase is individual productivity. This sounds modest, but it is not: across a team of twenty developers, consistent individual productivity gains compound into significant delivery acceleration at the project level. More importantly, it builds the developer trust that later phases depend on. Developers who have experienced Bob working well on their own tasks will engage constructively with the Walk phase. Those who were told to use a tool they never tried first will resist it.

> **Tip:** Keep the Crawl phase short — three to four weeks per cohort. Longer than that and teams start building habits and workarounds that the Walk phase will need to undo.

Phase 2 — Walk: Enterprise distributed environment

What changes in Walk

The Walk phase is where individual productivity becomes team capability. The shift is architectural: instead of every developer running their own Bob configuration against their own environment, the organization establishes a shared, version-controlled Bob setup that everyone on the project clones and uses.

This is the phase where governance decisions get made. What skills apply to this project? What rules enforce the coding standards? Which SAP systems does the project connect to, and how are those connections managed? Who controls the central repository, and what is the process for proposing changes to shared skills and rules?

Getting these decisions right in the Walk phase is what makes the Run phase scalable. Getting them wrong — or skipping them — creates a situation where every team has a slightly different Bob configuration, skills contradict each other, and no one trusts the output because no one agrees on what the standards are.

The central repository model

The architectural centerpiece of the Walk phase is a central Git repository that contains the project's Bob configuration: the MCP server definitions, the skills, the modes, the rules, and the environment structure. Every developer clones this repository. The connectivity details for all SAP and non-SAP systems live there — host names, system IDs, client numbers, API endpoints. What does not live there is credentials. Passwords, tokens, and client secrets belong in each developer's local environment file, which the repository's `.gitignore` explicitly excludes from tracking.

This separation is deliberate and enforced, not advisory. Bob cannot commit changes to the environment file. The `.gitignore` rules are locked. The reason is simple: a credential accidentally committed to a shared repository creates a security incident that will be remembered long after the productivity gains are forgotten.

Consider a project with a typical mid-size SAP landscape: two S/4HANA systems, one SuccessFactors tenant, one SAP Integration Suite, one BTP subaccount. Without the central repository model, each developer maintains their own connection details, diverging gradually as systems are renamed, IP addresses change, and new environments are added. The maintenance burden is real and the error rate is higher than teams admit. The central repository eliminates this. A developer who clones the project repository gets every connection configured correctly, immediately. Their environment file provides only their own credentials.

Skill separation by domain

A critical design decision in the Walk phase is how to organize skills. The instinct to create a single comprehensive skill that covers all SAP scenarios should be resisted firmly. Monolithic skills are hard to maintain, create conflicting guidance for developers working in different domains, and grow stale at the sections that are not actively used.

The right approach is to organize skills by SAP module and development domain, each in its own folder within the repository:

- A developer working on S/4HANA Sales gets the SD skills, with guidance on `VBAK`/`VBAP` structures, pricing procedures, and the relevant BAPIs and BADIs.
- A developer building a BTP CAP service gets the CAP skills, covering CDS model conventions, service handler patterns, and OData V4 configuration.
- A UI5 developer gets the Fiori and UI5 skills, covering XML view conventions, controller patterns, and `manifest.json` standards.

Each team gets a dedicated Bob mode that loads only the skills relevant to their work. This keeps context focused and prevents the model from receiving irrelevant guidance that degrades output quality. Version control means that skill changes go through a review process, just as code changes do. A team lead updating the SD skill to reflect a new client pricing rule creates a pull request. The change is reviewed, approved, and merged — and every developer picks it up automatically on their next pull.

Standardized project integrations

The Walk phase also formalizes the MCP integrations that the project depends on. In the Crawl phase, some developers may have configured experimental connections to JIRA, ServiceNow, or the SAP ADT layer. In the Walk phase, these connections are standardized, documented, and included in the central repository.

The project's MCP configuration specifies exactly which tools are available, which are auto-approved for use without developer confirmation, and which require explicit approval due to their write access or cost implications. This is governance at the tooling level — not policy documents that developers may or may not read, but actual configuration that controls what the agent can and cannot do.

> **Important:** SAP officially provides the ABAP Development Tools for VS Code extension, which includes a built-in ADT MCP server as part of SAP's supported toolset. As a result, community-driven MCP servers or third-party extensions are no longer required to enable AI-driven development. At the time of writing, connectivity uses standard SAP ADT APIs without custom RFCs. Be aware of the known limitations and ensure project's security review approves the selected connectivity approach before the Walk phase begins.

What Walk delivers

At the end of the Walk phase, the organization has something it did not have before: a reproducible, governable, version-controlled development environment that any new team member can be onboarded to in under an hour. The Bob configuration is not tribal knowledge living on a senior developer's laptop. It is an artifact, maintained like code, that the entire team shares and improves together.

The measurable outcomes are team-level: faster onboarding, fewer environment inconsistencies, higher consistency in code quality across developers, and a reduction in the time spent resolving "it works on my machine" discrepancies.

Phase 3 — Run: Intelligent orchestration at scale

What Run means in practice

The Run phase is where IBM Bob stops being a development tool and becomes an engineering platform — one that connects the full spectrum of an enterprise's delivery systems and operates with a level of automation that would not have been trusted before the Crawl and Walk phases built the necessary confidence.

In the Run phase, Bob does not wait to be asked. It participates in the delivery lifecycle: catching issues at the point of code authoring, triggering downstream processes when artifacts are created, connecting development activity to the change management and testing systems the enterprise already uses. The developer is still in the loop — always — but the loop is tighter, the handoffs are automated, and the cognitive overhead of switching between tools is dramatically reduced.

CI/CD integration and the role of IBM Bob as a gate

The Run phase integrates IBM Bob into the CI/CD pipeline. When a developer pushes code to the central repository, Bob participates in the review process: checking the change against the project's coding standards, running ATC checks against the SAP system, flagging violations before the change progresses toward transport.

This is not a replacement for human code review. It is a first pass that ensures human reviewers are spending their time on architectural and business logic questions, not on formatting violations or missing authorization checks. In practice, teams that implement this pattern report that the proportion of review comments addressing substantive issues — rather than mechanical standards — increases significantly.

The CI/CD architecture for SAP BTP deployments, illustrated in this publication's companion diagrams, shows IBM Bob positioned between the developer and the GitHub repository: reviewing code, approving or blocking the merge based on the project's rules, and triggering the SAP CI/CD webhook that initiates the build and deployment pipeline. What arrives at the SAP BTP Cloud Foundry environment has already passed Bob's review. The deployment pipeline then deploys the code. The developer gets a notification.

Cross-system orchestration

The Run phase unlocks cross-system workflows that were previously manual. A developer resolving a ServiceNow incident can ask Bob to read the incident, navigate to the relevant SAP code via the ADT connection, propose a fix, generate the technical design document, create the transport request, and update the ServiceNow ticket — all within a single session. Each step connects to a different system. Bob coordinates them.

This kind of orchestration is only reliable when the connections, the skills, and the guardrails were established carefully in the Walk phase. The reason the Run phase feels like flight rather than chaos is that the foundation was built correctly. Organizations that skip the Walk phase and try to orchestrate before their governance model is in place discover this the hard way.

Agentic execution and human oversight

A word on autonomy, because it is frequently misunderstood in discussions about AI-assisted development. The Run phase is not about removing humans from the loop. It is about repositioning humans in the loop at the points where human judgment genuinely matters.

Transport approvals, architectural decisions, customer-facing changes — these all involve human review. What the Run phase automates is the mechanical work that surrounds those decisions: gathering information, preparing artifacts, running checks, updating records. A senior developer who previously spent thirty percent of their time on administrative overhead gets that time back for the work that actually requires their expertise.

> **Important:** Spec-Driven Development amplifies the value of the Run phase significantly. Teams that invest in writing detailed specifications before asking Bob to act — defining goals, scope, edge cases, and acceptance criteria — produce consistently better outputs and use fewer tokens doing it. Vibe coding has its place in exploratory and prototype work. For production-grade SAP development, specifications come first.

Role transformation across the three phases

The three-phase framework changes not just tooling but roles. This is not a threat to existing roles — it is an evolution of what those roles spend their time doing, and the change is directional rather than abrupt.

A Business Analyst who once spent weeks translating requirements into functional specifications can, in the Run phase, review and refine specifications that Bob has drafted from structured input, spending their time on the ambiguities and edge cases that genuinely require human domain knowledge. An Integration Architect who once manually reviewed every mapping script now defines the patterns and guardrails that Bob applies consistently across the project. An ABAP Developer who once spent afternoons tracing `ST22` dumps can now describe the dump to Bob, get a diagnosis and a candidate fix, and spend their time validating the fix rather than hunting for it.

The roles are not disappearing. They are becoming more deliberate. The routine execution is increasingly shared with an agent that is very good at execution. The judgment, the context, and the accountability remain human.

Table: Role evolution across the three adoption phases.

Role	Crawl Phase Focus	Walk Phase Focus	Run Phase Focus
ABAP Developer	Personal prompting, code generation, debugging	Domain skills, ADT integration, team standards	Agentic workflows, transport automation, ATC gates
Integration Architect	CPI/Groovy generation, personal use	Shared integration patterns, MCP governance	End-to-end orchestration, CI/CD pipeline design
Project Manager	Awareness and support	Governance model, repository ownership	Delivery velocity metrics, onboarding acceleration
Enterprise Architect	Architecture validation	Skills and modes design	Platform governance, multi-team scaling
Senior Management	ROI framing	Progress against walk criteria	Platform cost model, strategic value realization

Governance foundations that enable scale

Sustaining the value of IBM Bob across a long engagement requires more than a good initial setup; it requires treating the governance layer as a living system. This section covers three foundations that determine whether the framework scales: keeping skills and rules current as living artifacts, maintaining the quality of context that Bob receives, and enforcing the security boundaries that protect the environment regardless of phase or pressure.

Skills and rules as living artifacts

The most common governance mistake in the Walk phase is treating skills and rules as one-time setup work. They are not. They are living artifacts that need the same discipline as code: version control, peer review, clear ownership, and a process for proposing changes.

A skill that is not updated when the client's business rules change produces guidance that contradicts the project's actual standards. A rule that was accurate at the start of a project but not revisited after a major SAP release may guide Bob toward patterns that are no longer valid. The organizations that sustain value from IBM Bob across a long engagement are those that assign explicit ownership of skills and rules, review them on a regular cadence, and treat requests to update them as normal development work — not administrative overhead.

The importance of context quality

Bob's output quality is a direct function of the quality of the context it receives. This is the central lesson of Spec-Driven Development, and it applies at every level of the adoption framework. In the Crawl phase, the developer is responsible for the context they provide in the prompt. In the Walk phase, the project team is responsible for the quality of the shared skills, rules, and environment configuration. In the Run phase, the organization is responsible for

maintaining the knowledge graph — the integration between Bob and the project's context management infrastructure — that gives the agent the information it needs to act reliably.

Context is not a one-time investment. It decays. Systems change, standards evolve, team knowledge accumulates. The organizations that realize the most value from IBM Bob are those that treat context maintenance as a first-class engineering discipline, not as something that happens automatically.

Security boundaries that cannot be crossed

Two security rules are non-negotiable regardless of phase. First: credentials never enter the central repository. Environment files are local, personal, and excluded from version control at the configuration level, not by convention. Second: Bob does not commit changes to environment files. This is enforced in the repository configuration, not requested of developers.

These constraints are not bureaucratic caution. They reflect real risk. A developer working quickly, under pressure, can make a mistake. The governance model exists to prevent that mistake from becoming a security incident. The cleaner and more automatic the enforcement, the lower the cognitive burden on the developer and the more reliably the constraint is observed.

Measuring progress across phases

Adoption progress should be measured, not assumed. The following indicators serve as phase transition criteria — evidence that the organization is ready to advance rather than a checklist to be checked off as quickly as possible.

The following transition indicators demonstrate readiness to move from Crawl to Walk:

- Developers can describe, unprompted, which Bob capabilities they use regularly and why.
- Average time-to-resolution on routine debugging tasks has measurably decreased.
- At least one domain-specific skill has been created from actual project requirements, not from a template.

The following transition indicators demonstrate readiness to move from Walk to Run:

- The central repository is the single source of truth for Bob configuration — no developer maintains a local override that differs materially from the shared setup.
- New team members reach full productivity with Bob within their first week.
- At least one cross-system workflow (e.g., JIRA ticket → code change → transport request) has been demonstrated end-to-end.

The following indicators demonstrate Run phase maturity:

- CI/CD pipeline includes Bob as a gate, and the gate is respected — builds that fail Bob's review do not proceed.
- Skills are reviewed and updated on a defined cadence, with ownership assigned.
- Senior leadership can point to concrete delivery outcomes — velocity, quality, onboarding time — that are attributable to the adoption program.

Summary and key takeaways

AI adoption in enterprise SAP environments does not succeed by accident, and it does not succeed by deploying a tool and declaring victory. It succeeds when the organization treats adoption as a deliberate program with a clear sequence, measurable outcomes, and governance that scales.

The Crawl, Walk, Run framework gives that program a shape. Start with individual productivity — let every person on the team experience Bob working well for them, in their own work, before asking them to contribute to shared infrastructure. Build the Walk phase carefully, because the quality of the central repository, the skill design, and the MCP governance model are what the Run phase stands on. Then, and only then, move to the orchestration and automation that make the investment genuinely transformative.

Three things make the difference between organizations that stall after Crawl and those that reach Run:

- **Context quality is an engineering discipline**: Skills, rules, and shared knowledge must be owned, maintained, and improved like code. They are not configuration files — they are the project's institutional knowledge, encoded.
- **Spec before code**: In production SAP development, the investment in a clear specification before asking Bob to act pays for itself in output quality, token efficiency, and reduced rework. Vibe coding has its place at the prototype stage. Enterprise delivery requires the discipline of specification-first.
- **Human oversight is not a compromise**: The goal of the Run phase is not to remove humans from the delivery process. It is to position human judgment where it is irreplaceable — at the decisions that require domain expertise, business context, and accountability — and to free it from the mechanical work that Bob handles well.

The sections that follow move from this strategic framework into the specific integrations, configurations, and development patterns that put these principles into practice.

Table: Consolidated phase summary for quick reference.

Dimension	Crawl	Walk	Run
Scope	Individual developer	Project team	Organization / program
Infrastructure	Local workstation	Central Git repository, shared MCP config	Full CI/CD integration, cross-system orchestration
Governance	Personal prompting habits	Version-controlled skills, rules, modes	Automated gates, context management, policy enforcement
Primary output	Individual productivity	Team consistency and onboarding speed	Delivery velocity at scale
Transition signal	Developer trusts Bob for routine tasks	New members onboard in under one week	CI/CD gate is respected; skills maintained on cadence

Governance and scale

Through the journey, you have gone through details on how IBM Bob can assist and provide you with an advantage when it comes to SAP engagements either transformation or operations based. This section mainly focuses on the key dimensions to consider for establishing the right governance and being able to deliver at scale using IBM Bob for your delivery programs within SAP.

This section starts by discussing about the guardrails, security posture and about human-in-the-loop design followed by practical framework for measuring value. It then closes by discussing the call to action about what you, your team and your clients should think about in the next 30, 60 and 90 days to turn the different scenarios covered in this publication to delivery mode.

Being able to deliver at scale with well-defined governance, needs a fundamental shift in thought process and every innovation along with radical shift arrives at the same turning point – the moment when enthusiasm meets accountability. That said, the promise of AI-assisted SAP delivery is real enabling the teams to move faster, decisions being well informed and the reduced cost of rework. These benefits are always followed by two questions to address:

- How do you know IBM Bob is doing the right thing?
- How do you know IBM Bob is actually working with the type of output needed?

To answer each of these questions, you need to first consider IBM Bob as a trusted partner rather than an autonomous solution that is unmonitored. The following sections provide the details.

Guardrails, security and human-in-the-loop

First, take an in-depth look at guardrails to understand why they are important and what they mean in practice.

Why is it important to have proper governance and guardrails in place?

In typical AI driven engagements, things usually get started with a quick proof of concept (POC) or pilot followed by validation with limited scenarios, which runs brilliantly getting everyone excited and ready to accept the solution. Then comes the pressure part to scale the solution quickly and findings related to scalability and integration related limitations coming into play along with not being able to fully control the changes to data or from an access point of view.

For example, a critical configuration recommendation is accepted without proper review by the team members, a sensitive dataset is exposed through an over-permissioned integration, or an automated workflow makes a decision that nobody anticipated. The key takeaway, here being – it is not to say adopting AI is incorrect but to say having the right guardrails from the get-go plays a key role in success of the approach.

What guardrails actually mean in practice?

The word "guardrail" gets used loosely in AI conversations, but in the context of SAP project delivery especially with IBM Bob, this word refers to a layered set of controls that govern what IBM Bob can access, what it can recommend, what it can execute autonomously and what must always be reviewed by humans before it takes effect.

These layers are categorized into four sections.

Section 1: Data access boundaries

IBM Bob operates within the data perimeter that you as an individual or the project teams define. It is not just about a matter of authorization that a user has in the system but also about establishing a clear data related domain during the deployment cycle. For example, project management data, SAP system configuration data, financial master data, human resources (HR) records and third-party integration payloads each carry different sensitivity levels.

It is important to define the access for different types of data explicitly and ensuring the same is well documented and auditable, in terms of who and what and how is the data being managed or processed. The success depends on evaluating this ahead in the process and not post go live. This is applicable not just for IBM Bob but also for AI projects in particular.

Section 2: Action boundaries

As described in Working with IBM Bob, under IBM Bob modes: Three personas, one tool, having a complete overview of the modes available and their impact is the key to having the right boundary set in terms of what IBM Bob can perform especially when it comes to what does IBM Bob recommend as an action versus IBM Bob executing the action. In Ask/plan mode, IBM Bob surfaces insights, drafts content, identifies risks and generates the plan of action needed. Similarly, in well-defined execution modes such as Agent mode, IBM Bob can take defined actions within SAP systems or surrounding ecosystem, but within the action boundaries you can define with custom mode being set. For information about how to create a custom mode, see Working with IBM Bob, under Step 4: Define the mode in custom_modes.yaml.

Think of it like a contractor on a construction site. A skilled contractor does not start knocking down walls the moment they arrive. They work to an agreed scope, seek sign-off before structural changes, and document everything they touch. IBM Bob operates the same way. The boundaries are not limitations on the capability of IBM Bob — they are the structure that makes the capability of IBM Bob safe to use.

> **Note:** A real-world example -

> During a large-scale S/4HANA Finance transformation, the project team configured IBM Bob to operate in 'advisory only' – plan mode (custom mode created) for the first eight weeks. Bob surfaced over 140 configuration recommendations, 23 of which were escalated for human review because they touched fiscal year variant settings that had downstream payroll implications. None of those 23 changes were auto-applied. Three were ultimately rejected by the finance lead. The guardrail worked exactly as intended.

Section 3: Model behavior

In addition to Data and Action boundaries, IBM Bob operates under behavioral constraints that govern how it reasons and communicates. These include hallucination-reduction mechanisms – where IBM Bob is anchored to verified SAP documentation, client-specific knowledge and confirmed project artifacts rather than generating unconstrained responses. This is possible either through a knowledge management solution based Model Context Protocol (MCP), such as using IBM Consulting Advantage based Context Studio as an input for the reference documentation. The approach is to also include the confidence thresholds, where IBM Bob explicitly can signal when it is operating at the edge of its knowledge or when a question requires human expert judgment.

This matters enormously in the SAP context, where configuration decisions can cascade across the entire SAP landscape. A confident-sounding wrong answer is considerably more dangerous than an honest "I am not certain – here is what I do know, and here is who you should ask".

Section 4: Audit and explainability

Every recommendation provided by IBM Bob is logged with its reasoning chain, for example – the logic it applied, and the confidence level it assigned. This is not just a compliance feature, it is a practical tool for project teams. When a recommendation is challenged six months later during a quality gate review, the team can trace back exactly how that recommendation was made and using which source of information. When something goes wrong, the audit trail supports fact driven postmortems rather than ambiguous questions about who made the decision.

Governance layers

Each of the previous four sections must be applied across different layers. This was discussed in Working with IBM Bob, under Centralized IBM Bob configuration and distribution for SAP teams:

- **Top governance layer**: Loads unconditionally, for every developer, on every session
- **Skill layer**: Loads dynamically, when trigger keywords relevant to a skill appear in a prompt
- **Connectivity layer**: Defines how IBM Bob reaches SAP and non-SAP systems, without encoding the secrets that authenticate to them
- **Environment variable layer**: Carries the non-secret configuration values that an installer or workspace needs
- **User layer**: Exists only on the developer's local machine and is never pushed to the repository

Consider the following examples:

- Advanced Business Application Programming (ABAP) rules: All custom object naming convention, transport request details being captured, authority checks to be performed for sensitive data checks.
- Every object handled needs a proper header and if already existing, the header needs to be updated with details such as – change request number, developer, transport, date and a running modification log.
- Custom code reaches for existing customized objects existing before creating a new one and even looking at the latest released Core Data Services (CDS) application programming interfaces (APIs) and ABAP RESTful Application Programming Model (RAP) patterns before it touches a table directly.
- There are no hardcoded business values, selects pull only the fields they need with a sensible `WHERE` clause, and database changes go through array operations rather than row-by-row loops.
- Sensitive operations carry authorization checks, updates respect the SAP lock concept, and deletion flags such as `LOVEM` and `LVORM` are honored instead of quietly ignored.
- Do not assume for documentation, be specific clear on the sources and maintain traceability and show only documentation or information crossing an 80% confidence threshold.

- Do not bypass authorization checks for business actions or configuration.
- Do not include secrets, passwords or production credentials in source code.
- Require peer review before transporting changes to a quality system.

And then there are the hard stops – the rules that exist because someone, somewhere, learned the lessons the expensive way. An example being, IBM Bob will not issue a `COMMIT WORK` inside an enhancement or a business add-in (BADI).

> **Note:** Why this matters for delivery
>
> A guardrail that lives in the workspace is auditable. You can show a security reviewer the exact standard that was enforced, point to the header on every generated object, and demonstrate that Clean Core was a gate rather than a hope. That is a very different conversation from "the AI is usually careful."

Benefits gained by different stakeholders

A well-defined governance and guardrails framework provides distinct advantages across key stakeholder roles:

- **Developers**: Get consistent and automatically enforced standards for naming conventions, structure of the output, documentation and by removing any ambiguity about what "correct" should be in a given scenario.
- **Architects**: Get a repetitive set of global rules for project-wide standards and domain-specific rules for shared technical policies along with mode-specific rules for per-technology enforcement without duplicating content across different projects within an account team.
- **Project managers**: To gain confidence that AI-assisted output will meet the same transport, testing and documentation standards along with repetitive output as it would be when a developer provides the effort, reducing rework and quality-gate failures later in the project cycle.
- **Senior management**: To gain an auditable governance layer given the rules are version-controlled, owned by named teams and enforced automatically on every interaction, supporting compliance and change-management objectives as AI-assisted development scales up.

The role of security

In alignment with guardrails and the approach defined previously, it is also important to have the right security architecture and approach in place. Take an in-depth look at the security aspect in terms of IBM Bob usage in an enterprise in the context of SAP.

One of the most common concerns project teams raise when IBM Bob is used – where does the data go and what is accessed?

Code quality is the visible half of the problem; the quieter half is access. An SAP landscape is full of things you do not want leaking into a model's context or, worse, into a document that gets mailed around: connection strings, service keys, bearer tokens for the very MCP servers that make IBM Bob useful. The discipline here is old-fashioned and has nothing to do with IBM Bob. Secrets stay out of tracked files and IBM Bob is pointed at an endpoint and a context identifier, never at a credential pasted into a prompt.

Another key aspect of security to consider with IBM Bob is to make sure you provide it the least-privilege thinking, similar to how you would operate with any other technical user. IBM Bob would read the data broadly but write narrowly, into development and into local or clearly named packages but never straight into anything transportable into production without a human-in-the-loop.

Although the preceding information deals with how you manage it with least-privilege code and also the way the code is deployed, you also need to consider how the code or data processed by IBM Bob is handled in terms of software as a service (SaaS) or Dedicated and other deployment models.

IBM Bob has different deployment options, both in regional instances of US, Europe, Middle East, and Africa (EMEA) and in Asia-Pacific (APAC) as well, along with an on-prem type deployment model that goes behind the firewall for specific client requirement if any. IBM Bob is deployed within the IBM enterprise-grade cloud infrastructure, which operates under International Organization for Standardization (ISO) 27001, System and Organization Controls (SOC) 2 Type II and where applicable, sector-specific compliance frameworks. For clients with data residency requirements, IBM Bob can be deployed in region-specific environments as mentioned previously and for clients with heightened security postures – defense, financial services, healthcare – private cloud and on-premises deployment options exist.

Every IBM Bob deployment is underpinned by the following key security design principles:

- **Data isolation**: Client data is never used to train shared models. Your SAP landscape, your project documentation, your configuration decisions – none of it leaves your tenancy.
- **Zero-trust connectivity**: Integrations between IBM Bob and SAP systems, whether it is SAP S/4HANA, SAP ERP Central Component (ECC), SAP Business Technology Platform (BTP), SAP Solution Manager or SAP Cloud Application Lifecycle Management (Cloud ALM) – are established through authenticated, encrypted least-privilege API connections. IBM Bob does not have standing access to production systems unless explicitly granted and monitored both outside through ABAP Development Tools (ADT), file system (FS), or MCP-based approach, as described in Connecting IBM Bob to SAP, under SAP ADT MCP server: The intelligent gateway.
- **Identity and access management**: IBM Bob respects the client's existing identity framework given each user interacts with IBM Bob based on their individual credentials and the actions of IBM Bob are attributed to those users in system logs within the SAP system.
- **Secrets management**: API keys, service credentials and integration tokens are managed through enterprise secrets vaults – never hardcoded, never embedded in prompts or conversation history.

As part of the SAP projects, it is strongly recommended to consider the preceding guidelines and design principles as part of the initial onboarding of IBM Bob and not after the fact, when IBM Bob is already active in the landscape. You can always consider the IBM security and compliance baseline assessment that maps the integration points of IBM Bob against the client's existing security controls and identifies any gaps before go live.

> **Note:** Chief information security officer (CISO) perspective -
>
> A global consumer goods company required that all AI tooling on their SAP program pass their internal Cyber Risk Assessment before go live. The security documentation for IBM Bob — including architecture diagrams, data flow maps, and third-party penetration test results — reduced the assessment timeline from the typical 12 weeks to just under five. The CISO's comment: 'This is the first AI vendor that came in already speaking our language.'

Human-in-the-loop: Driven by project teams, assisted by IBM Bob

Having guardrails and security design is a key part of the adoption model, but having human-in-the-loop is the core of the approach to make sure the project teams and the client are in control of the output being consumed.

The concept of "human-in-the-loop" has become the key and core expectation in all of the AI governance and project expectations, often as a reassurance mechanism without disrupting the flow of the value add from AI based solutions. In the context of IBM Bob and SAP delivery, it has a very specific and practical definition – there are certain categories and set of actions you get to choose on when IBM Bob should go ahead and auto process versus when you want IBM Bob to surface the analysis and wait for a human to make the decision providing explicit approval.

The question – how do you decide for which kind of activities and steps do you want auto processing from IBM Bob and for which ones do you want IBM Bob to pause and wait for human to act upon – this is not arbitrary because certain decisions have a material consequence not only for the project but for the business and for all stakeholders involved.

Consider a few of these categories.

Configuration decisions with downstream impact

In SAP implementations, some of the configuration choices can look minor but carry significant business process implications downstream for dependent processes. For example, posting period variants, document splitting rules, tax determination procedures and plant maintenance strategies are all examples of settings where an incorrect choice can create problems that take months to unwind. IBM Bob flags these decisions explicitly, presenting the options with their tradeoffs rather than quietly selecting an option provided first-hand. You can embed this into the custom mode option available to ensure IBM Bob provides you with options for a given task rather than going with the first decision coming out.

Code handling for deletion or complex updates

When processing SAP code either for ABAP or for advanced coding principles, it is key to being able to identify existing objects rather than generating new workflow, reports, interfaces, conversions, enhancements, and forms (WRICEF) objects from scratch. Hence, the rules and approach should be aligned to first identify the relevant object before going ahead with the code development / enhancement. Post the code generation, the checkpoints and quality gates need to be in place to confirm the right object is identified along with the code associated with the client requirement also validating the code usage.

Escalation and issue resolution

When IBM Bob identifies a project issue – a schedule risk, a scope creep pattern, a testing defect trend, it does not resolve it unilaterally. IBM Bob raises the issue along with the supporting evidence and suggests the options for resolution and routes it to the concerned owner. The project manager in this context remains in control of the escalation path and IBM Bob makes sure nothing gets ignored while ensuring the responsibility of the final decision to be made is still with experienced delivery team members.

Stakeholder-facing communications

Although IBM Bob can draft executive status updates, steering committee packs and change impact communications with impressive quality and time taken, the drafts are always expected to be reviewed by the team members before they get published. The human still remains accountable for what is going to be communicated, and the artifacts of IBM Bob are to support and expedite the time taken to generate the same and it is the start of the overall process, not the end point.

People decisions

IBM Bob provides insights and inputs to perform the resource planning, skills matching, and team capacity modeling. IBM Bob will never make or recommend a personnel decision without a human reviewing the context in full. It is recommended to process staffing, performance and team structure related decisions always with the humans to make the final decision. It is not just about being ethical but to be practical because IBM Bob cannot provide relational context that is typically needed for the right decision to be made.

The following table shows some examples of the possible risk tier and where the human role should be assessed:

Risk tier	Typical work	Human role
Low	Querying the ontology, reading CDS views, drafting a first-cut spec, generating a deck	Spot-check. Let it run; sample the output.
Medium	New reports, helper classes, test data, reverse-engineered technical specification documents (TSDs) in a dev package	Review before activation. Standard peer review applies.
High	Pricing, tax, credit, anything touching live order-to-cash or finance postings	Mandatory sign-off by a named owner. No exceptions, no shortcuts.
Hard stop	COMMIT in enhancements, production writes, secrets in code	Blocked outright. The assistant will not do it.

The practical effect of well-designed human-in-the-loop architecture is that teams feel empowered rather than displaced. When IBM Bob handles the high-volume, low-ambiguity work, status aggregation, document cross-referencing, defect triage, meeting summarization – the experienced professionals have more time and bandwidth for the decisions that genuinely require human involvement.

> **Note:** Program Director, Global Logistics Firm -
>
> A program director on a complex SAP S/4HANA greenfield rollout described it this way: 'Before IBM Bob, I was spending roughly a third of my week just chasing status. Getting people to update logs, consolidating reports, cross-checking the plan against actuals. IBM Bob handles that now. My week looks completely different. I actually have time to manage the program instead of administering it.'

In summary, having guardrails across different categories along with the security and human-in-the-loop approach is a key aspect for delivery especially when dealing with the SAP landscape. The next sections provide an in-depth look at how to measure the value and related outcomes.

Measuring value and outcomes

As the famous saying goes – "If you cannot measure it, you cannot manage it and more importantly, you cannot defend it". Every technology investment that survives its first budget cycle does so because someone in the organization took the time to build a clear, credible picture of value.

What makes measurement in AI-assisted delivery genuinely challenging is that the value often accumulates in places that traditional project metrics do not look. The hours that were not spent in duplicative status meetings, the risks that were identified three weeks earlier than they would have been without AI analysis, or the configuration errors that were detected before they reached production rather than after. These are real, material savings – but need a deliberate measurement approach to capture these benefits. This section will focus on the approach to be considered for being able to measure value and the outcomes.

The value generation typically can be measured in three phases – immediate, project level and strategic. This ensures you can set realistic expectations, define the measurement effort and then being able to progress with the communication to the stakeholders with the right context.

Immediate

The first category of value is the most visible and the easiest to measure. These are the direct hours saved, the cycle times shortened and the rework avoided as a direct result of the involvement of IBM Bob in day-to-day project activities.

This phase includes the following key metrics:

- Optimized time to generate the first set of code directly consumable for production usage
- Reduced number of defects caught in testing versus defects that reached user acceptance testing (UAT) or production
- Optimized incident resolution cycle time for SAP support requests routed through IBM Bob
- Optimized effort for impact analysis or change identification of the impacted objects in SAP landscape for the requirements
- Optimized time to produce weekly status reports
- Optimized hours spent in project meetings (specifically, meetings where the premeeting summaries and agenda preparation from IBM Bob eliminate the need for full attendance or long meetings)
- Reduced email and document search time (measured through periodic consultant time-logging or pulse surveys)

These metrics can typically be baselined in the first two to three weeks of deployment and trended from the fourth week onward. These are the metrics that resonate strongly with delivery teams and project sponsors who live and deliver the day-to-day reality of the project.

> **Note:** Efficiency metric in practice -
>
> One midsize manufacturing client measured a 38% reduction in the time their functional consultants spent preparing for blueprint workshops after IBM Bob was introduced. IBM Bob was preloading relevant SAP standard process documentation, cross-referencing it against the client's business requirements, and surfacing the key decision points ahead of each session. Consultants arrived informed rather than still reading.

Project Level

The second phase captures the impact IBM Bob has on project level outcomes, the kind of metrics that appear in steering committee dashboards and that define whether a project is regarded as successful.

This phase includes the following key metrics:

- **Schedule performance**: Are milestones being hit? Is the project running ahead of, on, or behind schedule compared to baseline – and how does that compare to comparable projects without AI assistance?
- **Cost performance**: Is the project tracking within budget? Are there measurable reductions in change request volume – a strong proxy for requirements clarity and early risk identification?
- **Quality metrics**: What is the defect density per functional area? How does the first-time pass rate on configuration compare to project benchmarks?
- **Risk realization rate**: Of the risks that IBM Bob identified and flagged for management, how many materialized versus how many were mitigated before becoming an actual issue?
- **Stakeholder sentiment**: Measured through structured pulse surveys at defined intervals – are business stakeholders more confident, better informed and more engaged than they typically are at equivalent stages of an SAP project?
- **Human involvement and iterations**: How many iterations as a human are you expected to optimize the code compared to when a human does it without AI assistance?

You need to also look at how outcome-based measurements are considered to make the IBM Bob adoption successful. The following table is the kind of scorecard and mix of what the previous sections covered:

Outcome	What to measure	What "good" looks like
Speed	Cycle time from approved spec to activated, review-passed object	Days become hours for standard objects; the backlog actually shrinks.
Quality	Defects found in quality assurance (QA) or UAT per object; review rework rate	Fewer defects per object, not more; first-pass review approvals rise.
Consistency	Share of objects meeting the coding and Clean Core standard on first submission	Standards compliance stops being a review finding and becomes the default.
Cost	Effort per deliverable (spec, object, test set)	Senior time shifts from typing to deciding; throughput per person rises.
Risk	Audit findings; security exceptions; failed transports	No new categories of incident; traceability improves, not degrades.

This is the phase where the compounding nature of the value of IBM Bob becomes visible. Early risk identification in week four does not just avoid a problem in week four – it prevents the schedule slip, the rework cycle, the additional testing pass and the steering committee conversation that the unmitigated risk would have triggered in week twelve or later. Capturing that chain of avoided cost requires connecting the dots deliberately, but the data is available for teams to work on.

Strategic

The third phase is the hardest to measure precisely but the most important to articulate. This is the value IBM Bob creates at the level of the global project or organization – changes in capability, knowledge and competitive position that outlast the project itself.

This phase includes the following key metrics:

- **Knowledge retention**: How much institutional knowledge from the project is captured, structured and retrievable through IBM Bob post go live? What is the impact on onboarding time for new team members? What is the reduction in dependency on a small number of "irreplaceable" knowledge holders?
- **SAP capability maturity**: Are business users demonstrating higher levels of SAP proficiency six months after go live compared to equivalent projects? Is the hypercare period shorter? Are self-service resolution rates higher?
- **Speed of future change**: As the client embarks on subsequent phases or new SAP deployments, how much of the groundwork already set by IBM Bob carries forward? Is the second rollout meaningfully faster and cheaper than the first?
- **Innovation velocity**: Are teams using the analytical capability of IBM Bob not just to manage the present project, but to explore future possibilities – identifying process optimization opportunities, evaluating new SAP capabilities, and building the business case for continuous improvements?

Accounts that invest in phase 3 related outcomes, tend to see something interesting happen, the business case for the next investment ask provides itself. It is strongly recommended to have a detailed value dashboard, which you can use to track and monitor the value generated.

Building the "value" dashboard to measure value

A value dashboard is useful only if it is built before the go live and even for a full scope adoption of IBM Bob for SAP projects and reviewed regularly during the delivery and taken seriously by the stakeholders with authority to act on it. The following is a practical structure to build an "IBM Bob value dashboard" for an SAP project.

Step 1: Define your baseline before IBM Bob gets adopted

You cannot demonstrate improvement without a starting point. In the first few weeks before IBM Bob is deployed in a project, capture baseline measurements for each of the metrics you intend to track. This does not need to be exhaustive – five to eight well-chosen indicators will serve better than thirty poorly measured ones.

Common baseline data sources for SAP include practitioner time logs, project management tool reports (JIRA, ServiceNow, SAP Cloud ALM), meeting calendars, testing tool defect counts and a brief stakeholder sentiment survey. The baseline investment is typically three to four hours of a project analyst's time. The return on that investment, in terms of the credibility of your value story, is considerable.

Step 2: Choose metrics that resonate with the audience

Different stakeholders care about different things and expect different metrics. For example, the project manager wants schedule and cost performance, the chief information officer (CIO) wants risk posture and system quality, the chief financial officer (CFO) wants efficiency ratios and cost avoidance, and business stakeholders want to know whether the system will actually work for their people.

Build the value dashboard with distinct views for each audience, drawing from the same underlying data but surfaced in the language that each stakeholder actually uses. IBM Bob can help construct these views, given its functionality beyond just handling SAP work to also provide applications to monitor the key performance indicators (KPIs).

Step 3: Review cadence and accountability

Schedule a monthly value review – a regular working session that brings together the IBM delivery lead, the client project manager and at least one senior business stakeholder. The agenda needs to be consistent: review the dashboard, discuss what is working, address what is not, and agree on any adjustments to how IBM Bob is being used.

This is beyond measurement; it is to create a shared accountability structure that keeps both the IBM team and the client team focused on outcomes rather than activities. The best SAP projects are ones where both IBM and client teams are equally invested in the same success criteria. The value review makes that investment explicit.

> **Note:** Governance insight -
>
> An energy sector client introduced a quarterly 'IBM Bob Value Board' — a cross-functional review attended by the CIO, CFO, and program sponsor. Within two quarters, the conversation had shifted from 'Is the AI worth the cost?' to 'Where else should we be using this?' The data had done its work.

But is measuring the metrics enough and reporting on them? Measurement is only half the scope, and the other half is communication to translate the data into a narrative that can influence people's perception. For example, the communication should be more specific and not general. Saying "IBM Bob has saved us time on reporting" is forgettable, rather providing "IBM Bob reduced our weekly status reporting cycle from fourteen hours of manual consolidation to three hours of review and approval" or "IBM Bob reduced our development lifecycle by 40%" is memorable and credible.

The measures should also connect to business outcomes, not project mechanics. For example, "IBM Bob identified the risk that our data migration approach would not support the required period-end close timing, three weeks before we would have discovered it the hard way", is a story about protecting the business. You should also be honest about limitations, because IBM Bob is not infallible – there will be recommendations that need to be reviewed and rejected. There are going to be insights, IBM Bob is going to miss. A value story is also about how you can acknowledge these gaps and explain how the human-in-the-loop design detected and corrected it – is far more credible than one that presents AI as 100% accurate.

This leads into the next section, which covers the conclusion and call to action.

Conclusion and call to action

SAP projects are among the most complex, consequential and politically charged initiatives that an organization will ever undertake. They touch every corner of the business; they consume enormous resources and the expectations attached to them are rarely modest. The failure rate of large enterprise resource planning (ERP) implementations, despite the accumulated methodology, tooling and hard-won experience, remains stubbornly high. Not because the

technology does not work, but because the human dimensions of these programs – coordination, communication, decision-making under pressure and knowledge management across large and distributed teams consistently prove harder to get right than the technical complexities.

IBM Bob addresses this in an effective way, where IBM Bob does not just replace the project manager's judgment, or the functional consultant's expertise or the change manager's relationship with the business, IBM Bob augments them by removing the friction that slows them down, filling the gaps that slip through under pressure, and connecting the dots across project landscape that is too large and too fast-moving for any individual to hold fully in their project delivery ask.

The teams that will extract the most value from IBM Bob are not necessarily the ones with the most sophisticated AI strategy, but the ones that approach the adoption of IBM Bob with clarity of purpose, knowing what the problems to be solved are and what boundaries you are setting and which outcomes you are committed to measuring. These are the teams that bring the rest of the teams along, rather than deploying technology over the heads of the people who have to work with it, with no proper enablement or guardrails being set properly.

Think of it this way – You would not let a brilliant new hire push to production unsupervised in their first week and you would not ignore them either. You would give them the standards, show them the gates, review their early work closely and widen their autonomy as they earned it. IBM Bob is no different, with one happy takeaway – after you have encoded a standard into its guardrails, it never forgets and never has an off day.

But what do some of the client deployments have in common?

In some of the close client related IBM Bob deployments across multiple SAP projects and different client types, a consistent pattern emerges among the ones that deliver the most sustained value.

The following key patterns emerged across these deployments:

- These projects started with a clear use case, not a technology mandate. The best project teams begin by asking "what is the hardest, most time-consuming, most error-prone part of the project right now?" and then deploying IBM Bob to address that specifically.
- The project teams resist the temptation to activate every feature at once but lay out a plan of action to achieve easing out the adoption by also earning the trust incrementally with the delivery team, with the client team and with the larger stakeholder community.
- The projects also have invested in the integration layer, IBM Bob is most powerful when it is connected to the SAP systems where the work actually happens along with integration to the project management platform, the testing tool and the documentation repository along with application lifecycle management systems.
- The projects that have invested the time to establish clean, governed integrations see dramatically higher value than those that use IBM Bob in isolation.
- The projects designated an IBM Bob champion, someone as a Forward Deployed Engineer (FDE) and from the delivery team, who owns the adoption experience, monitors the usage patterns and gathers feedback to continuously look for new ways to deploy the full features from IBM Bob point of view while addressing the project challenges. It is not a full-time role but is a named responsibility. Without a champion and FDE, IBM Bob adoption starts drifting toward the outer edge of the project rather than becoming central to how the project operates.
- The projects measured from the start, both before the IBM Bob adoption starts to be able to drive compelling value stories, and review the metrics regularly, and communicate their progress in the language that the stakeholders actually can understand and validate.

Key known limitations as of 3Q 2026

Although broadly the IBM Bob features help you set up the project framework and delivery related to it, there are known issues with live integration with SAP when it comes to SAP released ADT tools or Eclipse along with also domain knowledge from SAP modules such as Extended Warehouse Management (EWM), Billing and Revenue Innovation Management (BRIM), Transportation Management (TM), Sanctioned Party List Screening (SPL) and other complex modules not widely known for IBM Bob from the point of view of a large language model (LLM).

For IBMers and our business partners working on complex solutions and deployments, always stay up to date with the latest from IBM Bob through the internal seismic reference materials.

Your 30-60-90 day action plan

You have already started your journey in this context by choosing to read this publication and this is just the beginning, not the end. What follows is a concrete, actionable plan for putting the ideas in this section and in this publication into your SAP client delivery engagements. This publication has been designed to give you a practical overview into both integration and ways of working in the context of SAP to cover the journey of what an actual project team needs to consider and adopt to make it successful for IBM Bob adoption and for client delivery.

The following action plan is beyond the "Crawl – Walk – Run" concepts already explained in Hands-on environment setup for governed SAP development with IBM Bob, under Adoption roadmap: Crawl, Walk, Run, to give you a detailed feature adoption journey in terms of silo usage, rules/guardrails, skills, followed by system integration and seamless lifecycle handling.

Days 1 to 30: Lay the foundation

In this phase, focus on clarity and preparation by completing the following actions:

- **Identify your highest-value use case**: Where is the most pain right now? Reporting? Risk management? Testing coordination? Knowledge management? Pick one.
- **Conduct the security and compliance baseline assessment** with the IBM team. Understand your data domains, define your access boundaries, and establish your action boundary framework.
- **Baseline your metrics**: Before IBM Bob goes live, measure the current state of the indicators you intend to track.
- **Designate your FDE and IBM Bob champion**: Name them, brief them and give them the space to own the adoption experience.
- **Run a detailed workshop** not as a technical training, but a conversation about what IBM Bob is, what it is not and what the team's role in working with it will be. Address the anxiety honestly because most of it evaporates when people understand the human-in-the-loop design.

Days 31 to 60: Deploy and learn

In the second month, go live on your chosen use cases and build the habit on usage:

- **Deploy IBM Bob on the priority use cases** with a defined pilot group. Keep the scope narrow and the feedback loop tight.
- **Hold weekly 30-minute retrospectives** with the pilot group. What worked? What did not? What should IBM Bob be doing differently?
- **Review your first month of metrics**: Document the delta from baseline, share the early results with project leadership, even if they are modest, the direction of travel matters.
- **Begin identifying your other use cases,** and as you progress, apply your learning from the initial use cases to the next.
- **Connect IBM Bob to your core systems** if not already done. An integrated IBM Bob is exponentially more valuable than an isolated one.

Days 61 to 90 and beyond: Expand and embed

In the third month, scale what is working and build for the long term and beyond:

- **Activate your additional use cases** with the full delivery team. IBM Bob should now be a visible, routine part of how the project operates.
- **Run your first value review** by bringing together the IBM delivery lead, the client project manager, and other senior stakeholders from business.
- **Update your security and governance documentation** to reflect any scope expansion. Guardrails in place by now should be expanding and a living document to be further enhanced.

- **Begin capturing the qualitative user stories** to document not only what worked well but also what did not work well in terms of the risks that were averted, the decisions that were made better, the moments where IBM Bob made a material difference. These become the narrative layer on top of your quantitative dashboard.
- **Plan your post go live IBM Bob strategy** in terms of how IBM Bob will support hypercare, how the knowledge that IBM Bob has accumulated during the project will be made available to the support team and what the roadmap is for continuous improvement.

In summary, no technology delivers its potential in a culture that is not ready to receive it. The single biggest determinant of whether IBM Bob succeeds on an SAP project is not the sophistication of the technical deployment, but it is whether the team genuinely engages with the technology deployment initiative. The engagement starts with leadership and when the project director and project manager start using IBM Bob, and talk about it in steering committees, and credit it publicly when it adds value, the team tends to follow.

It also requires an honest conversation about what IBM Bob can do and cannot do along with what it means for team member roles. The anxiety that AI assistance creates in experienced professionals is real and understandable. Show the functional consultant that IBM Bob handles the document consolidation so they can spend their time on the work that actually requires their expertise. Show the project manager that the risk flagging of IBM Bob makes them look better in front of the client, not redundant. Show the business analyst that IBM Bob speeds up on first-draft documentation, freeing them to focus on quality and stakeholder engagement rather than ending up spending time formatting the documents.

Finally, the promise of AI in SAP delivery with IBM Bob is not just about machine and AI running better SAP delivery projects than people, it is that people who work with AI and IBM Bob in this context will run better projects than people who do not use IBM Bob.

Closure

You chose this publication because you are responsible for an SAP project delivery, or you are about to be, or you are building the practice that will deliver them. You understand the stakes around what these projects cost, what they demand and what they mean to the organizations that undertake them.

IBM Bob is not a shortcut; it is not a way to do less or to replace the experience and commitment that great SAP delivery requires. It is a way to make that experience and commitment to go further and to amplify the things that skilled people do well and to reduce the friction that wears them down. The guardrails are real and the measurement frameworks are real. The results being generated on projects around the world right now are real. What remains is for your project to be added to this story.

The next great SAP delivery is not going to look like the last one. It is going to be faster, better, informed and more connected because the teams delivering it will have a partner that never sleeps, never loses the threads and is always ready with the next insight when they need it.

IBM Bob is your partner in this journey for you to embark upon your SAP delivery.

Key takeaways

- Guardrails are not a constraint on the value of IBM Bob – they are the foundation that makes the value of IBM Bob safe to realize.
- Security is a first-class concern, not an afterthought. Address it before deployment, not after.
- Human-in-the-loop design keeps experienced professionals in control of the decisions that matter most.
- Measuring value across three phases – immediate efficiency, project outcomes and strategic impact to give you a complete and defensible picture of the contribution of IBM Bob.
- The 30-60-90 day action plan is your starting point alongside the crawl-walk-run approach provided in this publication. Adapt it, own it and hold your team accountable to it.
- Culture is the multiplier and having an honest leadership engagement and honest conversation about what IBM Bob means for people's roles will determine whether the technology delivers its potential.

Abbreviations

Abbreviation	Expansion
ABAP	Advanced Business Application Programming
abapGit	Open-source Git client for ABAP (community-driven)
ADT	ABAP Development Tools
AI	Artificial Intelligence
ALM	Application Lifecycle Management
ALV	ABAP List Viewer
AMDP	ABAP Managed Database Procedure
APAC	Asia-Pacific
API	Application Programming Interface
APIM	API Management
ATC	ABAP Test Cockpit
ATP	Available to Promise (SAP availability check)
AWS	Amazon Web Services
BAdI (BADI)	Business Add-In
BAPI	Business Application Programming Interface
BAS	SAP Business Application Studio
BASIS	SAP Basis (technology platform and system administration)
BDC	Batch Data Communication
Bobalytics	IBM Bob usage analytics and cost visibility feature
Bobcoins	IBM Bob token-based pricing unit (≈ USD 0.50 per coin)
BPMN	Business Process Model and Notation
BPP	Business Process Procedure
BRF+	Business Rule Framework+

Abbreviation	Expansion
BRIM	Billing and Revenue Innovation Management
BTP	SAP Business Technology Platform
CAP	SAP Cloud Application Programming Model
CDS	Core Data Services
CFO	Chief Financial Officer
CI/CD	Continuous Integration / Continuous Delivery (Deployment)
CIG	Cloud Integration Gateway
CIO	Chief Information Officer
CISO	Chief Information Security Officer
CLI	Command-Line Interface
Cloud ALM	SAP Cloud Application Lifecycle Management
CMDB	Configuration Management Database
CO	Controlling (SAP module)
CPI	Cloud Platform Integration (SAP Integration Suite)
CR	Change Request
CRM	Customer Relationship Management
CRUD	Create, Read, Update, Delete
CTMS	Cloud Transport Management Service
CTO	Chief Technology Officer
CTS	Change and Transport System
DCL	Data Control Language
DDIC	(ABAP) Data Dictionary
DEV	Development (system / tier)

Abbreviation	Expansion
DevOps	Development and Operations (integrated software delivery practice)
DMN	Decision Model and Notation
DND	Drag and Drop
DPC	Data Provider Class (OData service implementation class in ABAP)
E2E	End-to-End
EA	Enterprise Architecture
EAM	Enterprise Architecture Management
ECC	SAP ERP Central Component
EDI	Electronic Data Interchange
EMEA	Europe, Middle East, and Africa
EML	Entity Manipulation Language
ERP	Enterprise Resource Planning
ESM	Enterprise Service Management
ETL	Extract, Transform, Load
EWM	Extended Warehouse Management (SAP module)
FDE	Forward Deployed Engineer
FI	Financial Accounting (SAP module)
FM	Function Module (SAP ABAP)
FS	File System (as in ABAP Remote FS Extension)
FSD	Functional Specification Document
GDPR	General Data Protection Regulation
GenAI	Generative Artificial Intelligence
GitHub	Git-based source code hosting and collaboration platform (GitHub.com / github.ibm.com)

Abbreviation	Expansion
GitLab	Git-based source code hosting and collaboration platform (GitLab)
GOS	Generic Object Services
Groovy	Apache Groovy scripting language (used in SAP Integration Suite mappings)
GUI	Graphical User Interface
HANA	High-performance Analytic Appliance
HR	Human Resources
HTML	HyperText Markup Language
HTTP / HTTPS	HyperText Transfer Protocol (Secure)
IAS	SAP Identity Authentication Service
IBM	International Business Machines
ICA	IBM Consulting Advantage
IDD	Interface Design Document
IDE	Integrated Development Environment
IDoc	Intermediate Document
IdP	Identity Provider
IP	Internet Protocol
ISO	International Organization for Standardization
ITDD	Integration Technical Design Document
iTerra	IBM Test Evidence Review and Readiness Assistant
ITSM	IT Service Management
JIRA	Atlassian Jira (issue and project tracking platform)
JQL	Jira Query Language
JSON	JavaScript Object Notation

Abbreviation	Expansion
JSON-RPC	JSON Remote Procedure Call
JTI	JWT ID (JSON Web Token identifier)
JWT	JSON Web Token
KPI	Key Performance Indicator
LeanIX	SAP LeanIX (cloud-native Enterprise Architecture Management platform)
LLM	Large Language Model
LTS	Long-Term Support
MCP	Model Context Protocol
MDG	Master Data Governance
Miro	Cloud-based collaborative whiteboard platform
MM	Materials Management (SAP module)
MSAL	Microsoft Authentication Library
mTLS	mutual Transport Layer Security
MTX	SAP CAP Multitenancy and Extensibility services
MVC	Model-View-Controller
MVP	Minimum Viable Product
NFR	Non-Functional Requirement
NLP	Natural Language Processing
OAuth	Open Authorization
OData	Open Data Protocol
OO	Object-Oriented (ABAP)
OpenUI5	Open-source version of SAPUI5 UI development toolkit
OS	Operating System

Abbreviation	Expansion
OSS	SAP Online Service System; also Open-Source Software
PAT	Personal Access Token
PDF	Portable Document Format
PI/PO	Process Integration / Process Orchestration
PMO	Project Management Office
POC	Proof of Concept
PP	Production Planning (SAP module)
PR	Pull Request
PRD	Production (system / tier)
PS	Project System (SAP module)
QA	Quality Assurance
QE	Quality Engineer
RACI	Responsible, Accountable, Consulted, Informed (decision matrix)
RAP	ABAP RESTful Application Programming Model
RBAC	Role-Based Access Control
REST	Representational State Transfer
RFC	Remote Function Call
RISE	RISE with SAP
ROI	Return on Investment
SaaS	Software as a Service
S/4HANA	SAP S/4HANA (ERP suite)
SAC	SAP Analytics Cloud
SAP	Systems, Applications, and Products in Data Processing

Abbreviation	Expansion
SAPUI5	SAP UI development toolkit for HTML5 (proprietary version of UI5)
SD	Sales and Distribution (SAP module)
SDD	Spec-Driven Development
SDK	Software Development Kit
SDLC	Software Development Life Cycle
SF	SAP SuccessFactors
SFTP	Secure File Transfer Protocol
SICF	SAP Internet Communication Framework
SIEM	Security Information and Event Management
SIT	System Integration Testing
SOC	Security Operations Centre
SOC	System and Organization Controls (audit reports, as in SOC 2 Type II)
SOAP	Simple Object Access Protocol
SOW	Statement of Work
SOX	Sarbanes-Oxley Act
SPL	Sanctioned Party List Screening
SQL	Structured Query Language
SRE	Site Reliability Engineer
SRM	Supplier Relationship Management
SSH	Secure Shell
SSL	Secure Sockets Layer
SSO	Single Sign-On
STDIO	Standard Input/Output (MCP local transport type)

Abbreviation	Expansion
TC	Transaction Code
TLS	Transport Layer Security
TM	Transportation Management (SAP module)
TS	TypeScript
TSD	Technical Specification Document
TTL	Time To Live
UAT	User Acceptance Testing
UI	User Interface
UI5 (SAPUI5 / OpenUI5)	SAP UI development toolkit for HTML5
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
UX	User Experience
VPN	Virtual Private Network
VS Code	Visual Studio Code
watsonx	IBM watsonx AI and data platform
WM	Warehouse Management (SAP module)
WRICEF	Workflow, Reports, Interfaces, Conversions, Enhancements, and Forms
WSDL	Web Services Description Language
XML	eXtensible Markup Language
XSLT	Extensible Stylesheet Language Transformations
XSUAA	SAP BTP Authorization and Trust Management service
YAML	YAML Ain't Markup Language

Additional Resources

This document complements but does not replace IBM product documentation. For detailed API references, complete configuration options, and the latest updates, consult the following additional resources:

IBM Documentation

- **IBM Bob MCP Server documentation** — Refer to the IBM Bob product documentation for the current `mcp.json` schema, tool approval workflow, and MCP configuration reference.
- **IBM Bob Git documentation** — Refer to the IBM Bob documentation for the current built-in Git tooling reference.
- **IBM Bob MCP Marketplace** — Available within IBM Bob Settings → MCP. Includes the GitHub MCP Server, ToscaCloud MCP Server, and other integrations referenced throughout this publication.

IBM Internal Documentation

- IBM Bob — Understanding MCP *(internal, IBM employees only)*
- IBM Bob — Using MCP in Bob *(internal, IBM employees only)*
- IBM Bob — MCP Server Transports *(internal, IBM employees only)*
- IBM Bob Shell install script *(internal, IBM employees only)*
- IBM Bob Seismic home *(internal, IBM employees and BP's with access only)*

SAP Documentation

- SAP Help Portal — SAP Continuous Integration & Delivery
- SAP Help Portal — SAP Signavio Process Manager API
- SAP Help Portal — SAP LeanIX APIs
- SAP Help Portal — Connect to the MCP Server for SAP BTP Administration
- SAP Help Portal — SAP BTP Service Keys — Creating Credentials
- SAP API Hub — SAP Cloud ALM API
- SAP API Hub — SAP Cloud ALM REST API — Project Management
- SAP API Hub
- SAP Cloud ALM Documentation

abapGit

- abapGit Official Documentation (User Guide)
- abapGit GitHub Repository
- abapGit project home

> **Note:** The abapGit tool referenced throughout this publication is the open-source community edition, owned and maintained by the SAP community. The abapGit documentation is distributed under the

> Creative Commons Attribution 4.0 International license. For support or to report issues, refer to the community resources at https://docs.abapgit.org/development-guide/.

ServiceNow

- ServiceNow JWT Authentication documentation
- ServiceNow Table API reference
- **iTerra Reference Application** — The complete source code for the `release-note-checker-node` application (IBM Test Evidence Review and Readiness Assistant) is available in the companion repository for this publication.

GitHub

- GitHub Personal Access Token documentation
- **GitHub MCP Server** — Available from the IBM Bob MCP Marketplace (Settings → MCP).

Atlassian (Jira and Confluence)

- Jira REST API Documentation
- Jira Query Language (JQL) Reference
- Atlassian API Token Management
- Atlassian MCP Server (@sooperset/mcp-atlassian)

Microsoft

- Microsoft Graph API Overview
- Microsoft Graph API — SharePoint Sites
- Microsoft Graph API — Search
- Microsoft Graph API Reference
- Azure App Registration Guide
- MSAL Node (Microsoft Authentication Library for Node.js)

Tricentis

- Tricentis Tosca Documentation
- Tricentis CI/CD Integration Guide
- **Tricentis AI Workspace** — Available through the Tricentis platform; see Tricentis documentation for access configuration.
- **ToscaCloud MCP Server** — Available via IBM Bob MCP Marketplace (ToscaCloudMcpServer). Search within IBM Bob Settings → MCP.

MCP (Model Context Protocol)

- MCP Official Documentation

- MCP Getting Started
- MCP SDK (@modelcontextprotocol/sdk) — the standard Node.js SDK for building MCP servers compatible with IBM Bob.
- MCP TypeScript SDK — GitHub
- MCP Official Reference Servers — GitHub
- MCP GitHub Organisation
- Playwright MCP Server — install via `@playwright/mcp` in IBM Bob's `mcp.json`.
- Playwright MCP Server — GitHub (Microsoft)
- Playwright Official Documentation
- npm — @playwright/mcp
- **signavio-mcp (npm)** — Available via npm; install with `npx signavio-mcp` or clone the source repository for local customization.
- JSON-RPC 2.0 Specification
- OAuth 2.0 Client Credentials Flow (RFC 6749)

MCP Server Registries

- MCP Market — largest registry with 48,000+ servers, categorised by type (Collaboration, Cloud, DevOps, Browser Automation, and more).
- MCP.so — community registry with broad coverage.
- Smithery — curated marketplace with quality ratings.
- npm — search `@modelcontextprotocol` or `mcp-server-*` for packages.

Zod (Schema Validation)

- Zod Documentation — TypeScript-first schema validation library used in custom MCP server implementations.

Spec-Kit (Spec-Driven Development)

- Spec-Kit CLI — GitHub — install with `uv tool install specify-cli --from git+https://github.com/github/spec-kit.git`
- Spec-Kit Documentation

IBM Context Studio

- ICA Context Studio — Hands-On Lab *(internal, IBM employees only)*

Analyst and Industry Research

- Gartner — Top 10 Strategic Technology Trends for 2025
- Gartner — Top Strategic Trends in Software Engineering for 2025 and Beyond
- Gartner — Market Guide for AI Governance Platforms

Appendix: Program example zdownload_abap_object.abap

Copy and paste this program code into a file locally, then add it to your project for use.

```
*&---------------------------------------------------------------------*
*& Report ZDOWNLOAD_ABAP_OBJECT
*&---------------------------------------------------------------------*
*&
*&---------------------------------------------------------------------*
REPORT zdownload_abap_object.

" Selection screen definition
TABLES tadir.

PARAMETERS p_path TYPE string LOWER CASE.

SELECT-OPTIONS s_objnam FOR tadir-obj_name.

AT SELECTION-SCREEN.
  IF
        sy-ucomm = 'ONLI'
     OR sy-ucomm = 'PRIN'
     OR sy-ucomm = 'SJOB'.

    IF p_path IS INITIAL.
      MESSAGE 'Please enter a download path' TYPE 'E'.
    ENDIF.

    IF s_objnam[] IS INITIAL.
      MESSAGE 'Please enter object names' TYPE 'E'.
    ENDIF.
  ENDIF.

AT SELECTION-SCREEN ON VALUE-REQUEST FOR p_path.
  cl_gui_frontend_services=>directory_browse( CHANGING   selected_folder      = p_path
                                              EXCEPTIONS cntl_error           = 1
                                                         error_no_gui         = 2
                                                         not_supported_by_gui = 3
                                                         OTHERS               = 4 ).

  IF sy-subrc <> 0.
    MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
  ENDIF.


CLASS lcl_application DEFINITION.
  PUBLIC SECTION.
    TYPES ty_r_sobj_name TYPE RANGE OF sobj_name.

    METHODS constructor.

    METHODS main IMPORTING iv_path       TYPE string
                           it_r_obj_name TYPE ty_r_sobj_name.

  PRIVATE SECTION.
    DATA mt_selected_objects TYPE TABLE OF tadir.
    DATA mt_serialized_data  TYPE TABLE OF zif_abapgit_objects=>ty_serialization.

    METHODS select_objects IMPORTING it_obj_name TYPE ty_r_sobj_name.
    METHODS serialize_objects.

    METHODS download_files
      IMPORTING iv_path TYPE string.
ENDCLASS.


CLASS lcl_serializer DEFINITION.
```

```
  PUBLIC SECTION.
    METHODS serialize IMPORTING is_tadir        TYPE zif_abapgit_definitions=>ty_tadir
                      RETURNING VALUE(rs_files) TYPE zif_abapgit_objects=>ty_serialization.
ENDCLASS.


CLASS lcl_file_downloader DEFINITION.
  PUBLIC SECTION.
    TYPES ty_serialized_data TYPE zif_abapgit_objects=>ty_serialization.
    TYPES tt_serialized_data TYPE TABLE OF zif_abapgit_objects=>ty_serialization.

    METHODS constructor    IMPORTING iv_download_path   TYPE string.
    METHODS download_files IMPORTING it_serialized_data TYPE tt_serialized_data.

  PRIVATE SECTION.
    DATA mv_download_path TYPE string.

    METHODS create_directory_if_needed IMPORTING iv_directory      TYPE string
                                       RETURNING VALUE(rv_success) TYPE abap_bool.

    METHODS download_file IMPORTING iv_path          TYPE string
                                    iv_data          TYPE xstring
                          RETURNING VALUE(rv_result) TYPE i.
ENDCLASS.


CLASS lcl_serializer IMPLEMENTATION.
  METHOD serialize.
    DATA lx_error       TYPE REF TO zcx_abapgit_exception.
    DATA ls_i18n_params TYPE zif_abapgit_definitions=>ty_i18n_params.

    ls_i18n_params-main_language      = sy-langu.
    ls_i18n_params-main_language_only = abap_false.

    rs_files-item-obj_type  = is_tadir-object.
    rs_files-item-obj_name  = is_tadir-obj_name.
    rs_files-item-devclass  = is_tadir-devclass.
    rs_files-item-srcsystem = is_tadir-srcsystem.
    " rs_files-item-abap_language_version = zif_abapgit_dot_abapgit=>c_abap_language_version-standard.

    TRY.
        rs_files = zcl_abapgit_objects=>serialize(
                       is_item        = rs_files-item
                       io_i18n_params = zcl_abapgit_i18n_params=>new( is_params = ls_i18n_params ) ).

      CATCH zcx_abapgit_exception INTO lx_error.
        WRITE: / 'Error serializing', is_tadir-obj_name, ':', lx_error->get_text( ).
        CLEAR rs_files.
    ENDTRY.
  ENDMETHOD.
ENDCLASS.


CLASS lcl_file_downloader IMPLEMENTATION.
  METHOD constructor.
    mv_download_path = iv_download_path.
  ENDMETHOD.

  METHOD download_files.
    DATA lv_fullpath TYPE string.
    DATA lv_result   TYPE i.

    LOOP AT it_serialized_data ASSIGNING FIELD-SYMBOL(<ls_data>) WHERE files IS NOT INITIAL.
      WRITE: / 'Processing object:', <ls_data>-item-obj_type, <ls_data>-item-obj_name.
```

```
    LOOP AT <ls_data>-files ASSIGNING FIELD-SYMBOL(<ls_file>).

      " Create full path for file
      lv_fullpath = |{ mv_download_path }\\{ <ls_file>-filename }|.

      " Download file
      lv_result = download_file( iv_path = lv_fullpath
                                 iv_data = <ls_file>-data ).

      IF lv_result = 0.
        WRITE: / '  File downloaded:', lv_fullpath.
      ELSE.
        WRITE: / '  Error downloading file:', lv_fullpath, 'Code:', lv_result.
      ENDIF.
    ENDLOOP.
  ENDLOOP.
ENDMETHOD.

METHOD create_directory_if_needed.
  DATA lv_exists TYPE abap_bool.
  DATA lv_rc     TYPE i.

  rv_success = abap_true.

  cl_gui_frontend_services=>directory_exist( EXPORTING  directory            = iv_directory
                                             RECEIVING  result               = lv_exists
                                             EXCEPTIONS cntl_error           = 1
                                                        error_no_gui         = 2
                                                        wrong_parameter      = 3
                                                        not_supported_by_gui = 4
                                                        OTHERS               = 5 ).

  IF sy-subrc = 0 AND lv_exists = abap_true.
    RETURN.
  ENDIF.

  cl_gui_frontend_services=>directory_create( EXPORTING  directory                = iv_directory
                                              CHANGING   rc                       = lv_rc
                                              EXCEPTIONS directory_create_failed  = 1
                                                         cntl_error               = 2
                                                         error_no_gui             = 3
                                                         directory_access_denied  = 4
                                                         directory_already_exists = 5
                                                         path_not_found           = 6
                                                         unknown_error            = 7
                                                         not_supported_by_gui     = 8
                                                         wrong_parameter          = 9
                                                         OTHERS                   = 10 ).

  IF sy-subrc <> 0 AND sy-subrc <> 5. " 5 = already exists is ok
    WRITE: / 'Error creating directory:', iv_directory, 'Code:', sy-subrc.
    rv_success = abap_false.
  ENDIF.
ENDMETHOD.

METHOD download_file.
  DATA lt_data       TYPE solix_tab.

  lt_data = cl_bcs_convert=>xstring_to_solix( iv_data ).

  cl_gui_frontend_services=>gui_download( EXPORTING  filename               = iv_path
                                                     filetype               = 'BIN'
                                                     bin_filesize           = xstrlen( iv_data )
```

```
                                          CHANGING   data_tab                = lt_data
                                          EXCEPTIONS file_write_error        = 1
                                                     no_batch                = 2
                                                     gui_refuse_filetransfer = 3
                                                     invalid_type            = 4
                                                     no_authority            = 5
                                                     unknown_error           = 6
                                                     header_not_allowed      = 7
                                                     separator_not_allowed   = 8
                                                     filesize_not_allowed    = 9
                                                     header_too_long         = 10
                                                     dp_error_create         = 11
                                                     dp_error_send           = 12
                                                     dp_error_write          = 13
                                                     unknown_dp_error        = 14
                                                     access_denied           = 15
                                                     dp_out_of_memory        = 16
                                                     disk_full               = 17
                                                     dp_timeout              = 18
                                                     file_not_found          = 19
                                                     dataprovider_exception  = 20
                                                     control_flush_error     = 21
                                                     not_supported_by_gui    = 22
                                                     error_no_gui            = 23
                                                     OTHERS                  = 24 ).

    rv_result = sy-subrc.
  ENDMETHOD.
ENDCLASS.


CLASS lcl_application IMPLEMENTATION.
  METHOD constructor.
    " Initialize if needed
  ENDMETHOD.

  METHOD main.
    select_objects( it_r_obj_name ).
    serialize_objects( ).
    download_files( iv_path ).
  ENDMETHOD.

  METHOD select_objects.
    SELECT object, obj_name, devclass, srcsystem
      FROM tadir
      INTO CORRESPONDING FIELDS OF TABLE @mt_selected_objects
      WHERE obj_name IN @it_obj_name.

    IF sy-subrc <> 0.
      MESSAGE 'No objects found with the specified criteria' TYPE 'S'.
    ENDIF.
  ENDMETHOD.

  METHOD serialize_objects.
    DATA lr_serializer TYPE REF TO lcl_serializer.
    DATA ls_tadir      TYPE zif_abapgit_definitions=>ty_tadir.
    DATA ls_serialized TYPE zif_abapgit_objects=>ty_serialization.

    lr_serializer = NEW #( ).

    LOOP AT mt_selected_objects ASSIGNING FIELD-SYMBOL(<ls_object>).
      MOVE-CORRESPONDING <ls_object> TO ls_tadir.
      ls_serialized = lr_serializer->serialize( ls_tadir ).
```

```
      IF ls_serialized IS NOT INITIAL AND ls_serialized-files IS NOT INITIAL.
        APPEND ls_serialized TO mt_serialized_data.
        WRITE: / 'Object serialized:', <ls_object>-object, <ls_object>-obj_name.
      ENDIF.
    ENDLOOP.
  ENDMETHOD.

  METHOD download_files.
    DATA lr_downloader TYPE REF TO lcl_file_downloader.

    lr_downloader = NEW #( iv_download_path = iv_path ).

    lr_downloader->download_files( mt_serialized_data ).
  ENDMETHOD.
ENDCLASS.

START-OF-SELECTION.
  DATA gr_app TYPE REF TO lcl_application.

  gr_app = NEW #( ).
  gr_app->main( iv_path       = p_path
                it_r_obj_name = s_objnam[] ).
```

Notices

This information was developed for products and services offered in the US. This material might be available from IBM® in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 United States of America

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Trademarks

The following terms are trademarks of the International Business Machines Corp.

- **IBM**, the IBM logo, and **IBM Bob** are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.
- **SAP**, **SAP S/4HANA**, **SAP ECC**, **SAP BTP**, **ABAP**, **SAP NetWeaver**, and other SAP products and services mentioned herein are trademarks or registered trademarks of SAP SE in Germany and other countries.
- Other company, product, or service names may be trademarks or service marks of their respective owners. Mention of third-party products, community extensions, or open-source utilities (such as abapGit) is for reference and informational purposes only and does not imply direct endorsement or ongoing vendor warranty.

Notice on content examples

Purpose of Examples and Illustrative Content

All examples, walkthroughs, demonstration scenarios, code snippets (including ABAP, JSON, XML, shell scripts, and configuration files), architectural diagrams, and integration patterns presented in this book are provided solely for **demonstration and educational purposes**.

- **Not Released or Production-Grade Product Code**: The code samples and scripts contained herein do not constitute officially released, commercially warranted, or production-ready software deliverables from IBM, SAP SE, or any other technology vendor.
- **User Responsibility and Due Diligence**: Any adoption, reproduction, adaptation, or execution of the code snippets, configuration scripts, or integration logic provided in this document is done entirely at your own discretion and risk. Organizations and individual practitioners must thoroughly evaluate, test, adapt, and validate all logic in accordance with their enterprise standards, security policies, transport procedures, and governance frameworks before deploying to non-production or production environments.

Anonymized Systems, Hostnames, and Identifiers

All system identifiers, application IDs, server hostnames, IP addresses, endpoint URLs, user accounts, organization names, and transport numbers referenced throughout this book are **entirely fictional and illustrative**.

- Any resemblance to actual live systems, internal enterprise networks, or active environments is purely coincidental.
- This publication does not provide or grant credentials, licenses, or access rights to any IBM, SAP, GitHub, or third-party infrastructure.
- To establish connections or acquire access to specific enterprise environments, repositories, or API endpoints, users and project teams must follow their organization's standard procurement, security, and administrative access-request procedures through the appropriate authorized owners.

Custom MCP Servers and Architectural Patterns

The custom Model Context Protocol (MCP) servers, connector wrappers, and middleware patterns demonstrated across the publication serve as **conceptual communication blueprints** designed to illustrate how AI assistants interact with enterprise backends.

- They are intended to build technical understanding and demonstrate possibility.

- When designing and implementing actual enterprise-grade integrations, architects and developers should apply the architectural concepts and principles discussed here while engineering solutions that adhere to their enterprise security baselines, network isolation boundaries, encryption standards, and maintenance requirements.

Point-in-Time Technical Validity and Software Evolution

The technical guidance, configuration parameters, API references, code syntax, and IBM Bob capabilities documented in this book reflect the software state, feature sets, and best practices available at the time of writing.

- **Version Context**: The demonstrations and walkthroughs in this publication were developed, tested, and validated using **IBM Bob Version 2.0.5**.
- **Evolving Tooling**: Software features, user interfaces, CLI commands, extensions, API endpoints, and platform capabilities evolve continuously across release cycles for IBM Bob, SAP S/4HANA, SAP BTP, Visual Studio Code, Git platforms, and open-source utilities like abapGit.
- Readers and implementation teams are strongly advised to consult the latest official product documentation, release notes, and vendor support channels to confirm current syntax, supported release levels, and compatibility requirements during real-world implementations.

AI Attribution

This IBM Redbooks publication was produced with the selective assistance of three AI tools: Claude Code, IBM Bob, and ICA Edge (IBM Consulting Advantage Edge). These tools were engaged for specific, bounded tasks — not for the creation of original ideas or technical content. Claude Code was used to reconcile chapter text across multiple draft sources and to apply IBM Redbook formatting conventions including headings, callout boxes, and layout structure. IBM Bob assisted with reviewing SAP-specific and industry terminology and formatting code blocks for consistency. ICA Edge was used to create and format architectural diagrams and workflow visuals used in the book. All AI-generated or AI-assisted output was reviewed, approved, and verified by the co-authors before inclusion. The ideas, architecture patterns, demonstration code, and technical conclusions expressed in this book are entirely the author's own, drawn from real-world IBM consulting and implementation experience (with few client feedback inclusion). There are certain limits in terms of connectivity along with domain experience, for which premium package work is in progress. AI tools served as a formatting and production layer only — not as a source of substance, analysis, or judgment.

The content, demo code, and concepts in this Redbook are written solely by the author. AI tools were used only for reconciliation, formatting, and diagram creation tasks. AI was used to make stylistic edits, such as changes to structure, wording, and clarity. AI was used to make content edits, such as changes to scope, information, and ideas. AI was prompted for its contributions, or AI assistance was enabled. AI-generated content was reviewed and approved.

The use of AI tools in producing this Redbook is intentional and transparent. The book is written on "Delivering SAP with IBM Bob" which is itself subject of this book — using it as part of the authoring process reflects the author's confidence in their production readiness for senior IBM practitioners.

Version History

Version 1.0

Document Number: MD260033

Title: Delivering SAP with IBM Bob

Authors:

- Dharma Atluri
- Tuhin Bhaduri
- Shubhangi Joshi
- Phillip Gerrard
- Meher Charan Vangipurapu
- Mohammad Zubair

Publication Date: October 7, 2026

Summary: Initial publication providing practical guidance for IBM consultants, SAP practitioners, and enterprise architects on using IBM Bob — IBM's AI-native engineering assistant — to accelerate and govern SAP development across the full software development lifecycle. Covers SAP connectivity approaches (ABAP Remote FS Extension, ABAP ADT Extension, and SAP ADT MCP Server), IBM Bob modes, permissions, Rules and Skills for enterprise governance, Clean Core standards enforcement, centralized team configuration, and spec-driven development with Spec-Kit. Addresses practical ABAP, BTP, UI5, CAP, and SAP Integration Suite development patterns, documentation generation (FSD, TSD, BPP, IDD), and deep coverage of IBM Bob's MCP integration ecosystem including ServiceNow, Jira, SharePoint, GitHub, Tricentis Tosca, SAP Signavio, SAP LeanIX, SAP Cloud ALM, and guidance on building custom MCP servers. Includes a hands-on getting started guide and a Crawl–Walk–Run enterprise adoption roadmap, and an appendix with a sample ABAP program for downloading ABAP objects.